Abstract

System integration is a key enabler to maximize information value in an engineering context. The valuable information is normally represented by information models which play a decisive role in the implementation of system integration. The information models are designed to efficiently and effectively capture, process and communicate information among different functional and business units. However, use of the information models in implementing system integration is challenged by insufficient support from current settings of modeling architectures. This situation calls for new strategies to ease the use of information models.

To address this challenge, this study presents a new twofold solution: Model driven system integration. It includes 1) a modeling architecture to guide the development of information models and 2) an integrated implementation process to guide the use of information models. Thus, this work improves practical applicability of an information model in its entire modeling lifecycle.

The results contribute not only to the performance of modeling practices but also to improved understanding of information modeling in system integration. Implementation contexts and implementation models are introduced to develop an implementation-oriented modeling architecture. Further, the potential of information models as a knowledge base to support implementation practices is identified.

To concretely discuss behaviors and structures of information models, this study adopts ISO 10303 and the related standards as major references of existing information models.

Case studies on model driven system integration validate this research in scenarios concerning kinematic modeling, kinematic error modeling, cutting tools classification and product catalogue modeling. Model driven system integration exhibits high efficiency in implementation, enhanced interoperability and increased value of information models.

Keywords
System architecture, system integration, information model, ISO 10303, application context, implementation context, implementation model.
Sammanfattning

Systemintegration är väsentligt när man vill maximera informationsvärdet i ingenjörsmässiga sammanhang. Värdefull information representeras normalt av informationsmodeller som spelar en avgörande roll i genomförandet av systemintegration. Dessa informationsmodeller är utformade för att effektivt fånga, bearbeta och förmedla information mellan olika funktionella enheter och affärsenheter. Emellertid stödjs inte användningen av informationsmodeller tillräckligt vid genomförandet av systemintegration i nuvarande modelleringsarkitekturer. Detta gör att integration försvåras eller inte blir av, vilket kräver nya strategier för att underlätta användningen av informationsmodeller.


För att konkret diskutera beteenden och strukturer av informationsmodeller, använder denna studie ISO 10303 och tillhörande standarder som viktiga referenser av befintliga modelleringsarkitekturer. Fallstudier inom modelldriven systemintegration används för validering, i scenarier för kinematisk modellering, modellering av kinematiska fel, klassificering av skärande verktyg och produktkatalogmodellering. Modelldriven systemintegration uppvisar hög effektivitet vid genomförandet, förbättrad interoperabilitet och ökat värde av informationsmodeller.
Acknowledgement

At the end of this graduate study, I am extremely proud to work with many colleagues and friends during my journey here. The dissertation would not happen without the unbelievable patience and wonderful support of my major supervisors, Prof. Torsten Kjellberg and Dr. Mikael Hedlind. They helped me great to better organize and formulate my ideas. Prof. Lars Mattsson and Dr. Gunilla Sivard are also great supervisors who have, throughout my study, mentored and supported me.

I am honored to work with many talented people offering enlightened advices and content to my study, including Dr. Danfang Chen (陈丹芳), Magnus Lundgren, Prof. Lihui Wang (王力翚), Dr. Rong Su (苏榕), Qiuling Huang (黄秋灵), Dr. Thomas Lundholm, Christer Wickman and Andreas Bergstrand. They added much to the depth and broadness of my research.

I would like to thank the cool Chinese fellows for their amazing support to my work and my living in Sweden, to whom I am greatly indebted: Dr. Wei Ji (计伟), Dr. Xu Liu (刘旭), Dr. Wei Wang (王伟), Dr. Dawei Liu (刘大炜), Dr. Qilin Fu (付麒麟), Hongyi Liu (刘弘逸), Bitao Yao (姚碧涛), Dr. Yue Meng (孟悦) and Dr. Vicent Wang (王曦).

I am grateful to many colleague “IIPers” for their support, including Assoc. Prof. Daniel T. Semere, Assoc. Prof. Lasse Wingård, Per Johansson, Navid Shariat Zadeh, Lars Lindberg, Dr. Antonio Maffei, Prof. Mauro Onori, Dr. João Ferreira, Johan Pettersson, Anna Wiippola, Sarah Golibari and Anna Eklund.

I would like to acknowledge Prof. Zhizhong Li (李志忠), Prof. Pei-Luen Patrick Rau (饶培伦) and Prof. Kaibo Wang (王凯波), from Tsinghua University, Beijing, who inspired me to be devoted to academic research.

I have been fortunate to have my beloved parents, who have been always there with extraordinary kindness and goodwill and my dear friends in China and America, Dr. Yan Deng (邓彦), Xiaolu Dong (董晓璐), Dr. Ji-yuan Gao (高基元), Prof. Ruiwei Jiang (江瑞威), Jianfeng Liao (廖剑锋), Xiaoting Lin (林晓婷), Fei Song (宋霏), Guanxiang Wan (万观祥), Rongwen Xie (谢榕文) and Prof. Ming Zeng (曾鸣).
Abbreviations

AAM  Application Activity Model
AEC  Architecture, Engineering and Construction
AIM  Application Integrated Model
AM   Application Module
AM MVC Application Model MVC
AP   Application Protocol
API  Application Programming Interface
ARM  Application Reference Model
ASD  Automotive, aerospace and defense industry
ASME American Society of Mechanical Engineers
BIM  Building Information Model
CAD  Computer-Aided Design
CAM  Computer-Aided Manufacturing
CAx  Computer-Aided technology
CNC  Computer Numerical Control
DFBB Digital Factory Building Blocks
GD&T Geometric Dimensioning and Tolerancing
GPS  Geometrical Product Specification
GUI  Graphical User Interface
FBOP Feature Based Operation Planning
HCD  Human-Centered Design
HCI  Human-Computer Interaction
IA   Information Architecture
IaaS Infrastructure as a Service
IDEFo Integrated computer aided manufacturing DEFinition for Function modeling
IDM  Information Delivery Manual
IEC  International Electrotechnical Commission
IEEE Institute of Electrical and Electronics Engineers
IFC  Industry Foundation Classes
Impl. Implementation
ISO  International Organization for Standardization
IT   Information Technology
IxD  Interaction Design
JAR  Java ARchive
LCIM Level of Conceptual Interoperability Model
Mfg. Manufacturing
MIM  Module Interpreted Model
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPQP</td>
<td>Model-driven Process and Quality Planning</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>MVD</td>
<td>Model View Definition</td>
</tr>
<tr>
<td>MVP</td>
<td>Model-View-Presenter</td>
</tr>
<tr>
<td>OOSE</td>
<td>Objective-Oriented Software Engineering</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PFMEA</td>
<td>Process Failure Mode and Effects Analysis</td>
</tr>
<tr>
<td>PLCS</td>
<td>Product Life Cycle Support</td>
</tr>
<tr>
<td>PLib</td>
<td>Parts Library (ISO 13584)</td>
</tr>
<tr>
<td>PLM</td>
<td>Product Lifecycle Management</td>
</tr>
<tr>
<td>PPR</td>
<td>Product-Process-Resource</td>
</tr>
<tr>
<td>RPN</td>
<td>Risk Priority Number</td>
</tr>
<tr>
<td>Saas</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SDAI</td>
<td>Standard Data Access Interface</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SoS</td>
<td>System of Systems</td>
</tr>
<tr>
<td>STEP</td>
<td>STandard for the Exchange of Product data (ISO 10303)</td>
</tr>
<tr>
<td>STEP-NC</td>
<td>NC programming language extending STEP (ISO 14649)</td>
</tr>
<tr>
<td>TTM</td>
<td>Time To Market</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>UoF</td>
<td>Unit of Functionality</td>
</tr>
<tr>
<td>UX</td>
<td>User Experiences</td>
</tr>
<tr>
<td>VBA</td>
<td>Visual Basic for Applications</td>
</tr>
<tr>
<td>XaaS</td>
<td>Everything as a Service</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XP</td>
<td>eXtreme Programming</td>
</tr>
</tbody>
</table>
List of appended publications

**Paper A**

**Paper B**

**Paper C**

**Paper D**

**Paper E**

**Paper F**
Authored publications not appended


Contents

Chapter 1  Introduction ......................................................................................... 1
  1.1 Research scope ............................................................................................. 2
    1.1.1 Model driven system integration ......................................................... 2
    1.1.2 Architcting .......................................................................................... 5
  1.2 A new modeling architecture ................................................................. 7
    1.2.1 Pragmatic concern for implementers ................................................. 7
    1.2.2 Development of a new modeling architecture.............................. 9
  1.3 An integrated implementation process ............................................... 13
    1.3.1 Two types of unused potential of information models.............. 13
    1.3.2 Make use of the potential in process integration .................... 15
  1.4 Main contributions ..................................................................................... 17
    1.4.1 Identification of the root cause of challenges ......................... 17
    1.4.2 Solutions ........................................................................................... 19
    1.4.3 New concepts ................................................................................... 20

Chapter 2  Frame of reference ......................................................................... 23
  2.1 Interoperability .......................................................................................... 23
    2.1.1 Interoperation as a challenge ......................................................... 24
    2.1.2 How to understand interoperability .............................................. 26
    2.1.3 Semiotic aspects of interoperability ............................................ 27
    2.1.4 Pragmatics: A focus of this study ............................................. 33
  2.2 Standardized information models ............................................................ 36
    2.2.1 Application contexts and information requirements ............ 37
    2.2.2 The STEP modeling architecture .............................................. 43
    2.2.3 Extensibility and portability of STEP ....................................... 49
    2.2.4 High extensibility is preferable .................................................... 52
  2.3 A model driven approach ......................................................................... 54
    2.3.1 Contextual proprietary information models .......................... 54
    2.3.2 Composable standardized generic information models ....... 56
    2.3.3 Combination of contextual and composable solutions ...... 57
  2.4 System architectures .................................................................................. 58
Figures and tables

Figure 1.1 A standardized contractor in system integration. .................. 3
Figure 1.2 Model driven system integration: An interdisciplinary topic. .... 4
Figure 1.3 Architectures and stakeholders in a context of the model driven system integration. ................................................................. 6
Figure 1.4 Information models to facilitate semantic interoperability.
Both data schemas and data sets are information models. ............... 7
Figure 1.5 Implementers interpret a part of modeling architecture to
develop applications conforming to the data schemas. ................. 8
Figure 1.6 An envisioned situation of model development and software
development .................................................................................................. 11
Figure 1.7 Publications relative to the layered architecture............. 12
Figure 1.8 Results of the complex modeling architectures .............. 14
Figure 1.9 Software implementation utilizes different types of information
models in a modeling architecture ................................................................. 15
Figure 1.10 An integrated process for model driven system integration.. 15
Figure 1.11 The conventional modeling architecture of ISO 10303 STEP,
based on textual description by Anderl and Wasmer (1997) ........ 18
Figure 1.12 Thesis structure ................................................................. 20
Figure 1.13 An application as an intermediate between stakeholders and
information models. ................................................................................. 21
Figure 2.1 Reuse of existing data by implementers and users ........ 25
Figure 2.2 A situation where programs (i.e. application software) control
users, presented by Richard Stallman at TED (2014) ....................... 26
Figure 2.3 “A sign-situation, or a process of semiosis” (drawing based on
Morris, 1939) ............................................................................................... 28
Figure 2.4 Four cornerstones and three connections of the framework
proposed by Lindland, et al. (1994) ......................................................... 29
Figure 2.5 Model based computerized information systems (reproduction
based on Hofmann, 2003) ........................................................................ 31
Figure 2.6 Three levels of interoperability concerning this study ........ 34
Figure 2.7 Computer interpretation of information models (data sets)
needs human interpretation of information models (data schemas).
......................................................................................................................... 35
Figure 2.8 Application context used in software implementation .... 38

XIX
Figure 2.9 The application contexts defining information models. .......38
Figure 2.10 A traditional view of information requirements as a subset of requirements to be implemented in information models...........40
Figure 2.11 A data schema and a data set..................................................40
Figure 2.12 Application context, the current basis for application development ................................................................................. 41
Figure 2.13 Application context, the current basis for both application development and information model development..................42
Figure 2.14 Application context used for information model development. ........................................................................................................ 43
Figure 2.15 The traditional modeling architecture of ISO 10303 STEP...44
Figure 2.16 Artifacts (rectangles) in STEP modeling architecture useful for implementers................................................................. 45
Figure 2.17 Software development based on the STEP modeling architecture..................................................................................46
Figure 2.18 A workflow to implement information model standards based on STEP modeling architecture........................................ 48
Figure 2.19 Information models from different domains are used by applications focusing on applying multiple views of product data..49
Figure 2.20 A data schema and a data set.......................................................50
Figure 2.21 Extending the data schema based on information requirements from new application context.................................51
Figure 2.22 Porting the information model to new application..............51
Figure 2.23 Reusable building blocks to compose an information model. .................................................................................................53
Figure 2.24 Contextual abstraction and composable abstraction..........56
Figure 3.1 Pragmatic interoperability that should be addressed by information models.................................................................66
Figure 3.2 Pragmatic interoperability of an information model facilitates portability, usability and implementability with implementers as direct users.................................................................68
Figure 3.3 The current situation: The application context forms a basis for application implementation as well as model development........69
Figure 3.4 The traditional way to implement an application conforming to a STEP data schema.................................................................70
Figure 3.5 Evolution of the modeling architectures, based on ISO (2016c).

Figure 3.6 The usage of the implementation context.

Figure 3.7 Implementation context, a shifted basis where information model development begins.

Figure 3.8 User groups and their positions in an implementation context.

Figure 3.9 Tasks performed with CADCAM systems, based on (Makris, et al. 2014; Zeid, 1991).

Figure 3.10 In the information model development process.

Figure 3.11 UI of Siemens NX 7.5 visualizes a kinematic link in relation to its corresponding geometry.

Figure 3.12 UI of STEP-NC Machine 9.38 visualizes motion simulation.

Figure 3.13 UI of a STEP AP242 model translator developed in this study.

Figure 3.14 Components in an implementation context.

Figure 3.15 Information requirements based on implementation contexts.

Figure 3.16 An example for instantiating a kinematic link, excerpted from an AP242 data set.

Figure 3.17 Diagrammatic information requirements for an implementation model for kinematic data exchange.

Figure 3.18 A class diagram for kinematics in the STEP Toolbox API for Java.

Figure 3.19 The layered modeling architecture.

Figure 3.20 A closed layered architecture.

Figure 3.21 An open layered architecture.

Figure 3.22 Decoupling to reduce dependency.

Figure 4.1 A traditional mapping process for using a data schema.

Figure 4.2 Current integration of the information model mapping with the implementation process.

Figure 4.3 Closely integrating the mapping process with the implementation process.

Figure 4.4 A general mapping pattern of engineering activities.

Figure 4.5 Models are used to translate reality to products.
Figure 4.6 Software implementation supports (physical) product realization in manufacturing industry: The scope of this chapter. 115
Figure 4.7 Apply data schemas for system integration. 116
Figure 4.8 Information models are used in implementation. 116
Figure 4.9 Information models have potential to support implementation. 117
Figure 4.10 Categorization of engineering design activities (McNeill, et al., 1998). 118
Figure 4.11 Segments of design protocols (Akin and Lin, 1995). 118
Figure 4.12 The waterfall software implementation style aimed at eliminating unwanted iterations (Royce, 1970). 119
Figure 4.13 The incremental development style and the prototyping development style. 120
Figure 4.14 Microsoft “Synch and stabilize” process (Cusumano and Selby, 1997). 122
Figure 4.15 The 4+1 view model of software architecture. 123
Figure 4.16 Four activities of interaction design (Sharps, et al., 2007). 124
Figure 4.17 The star model (Hix and Hartson, 1993). 125
Figure 4.18 The standardized HCD process (ISO, 2010b). 125
Figure 4.19 Usability engineering lifecycle (Mayhew, 1999). 126
Figure 4.20 A generalized process of software implementation. 127
Figure 4.21 Processes need to be integrated. 128
Figure 4.22 The information model mapping process underpin a process to implement software at different levels of abstraction. 129
Figure 4.23 Integration of the processes. 130
Figure 4.24 Stages and activities of the implementation process. 131
Figure 4.25 Activities of Model driven software implementation process for system integration and the parallel processes. 133
Figure 4.26 The highest level of the activity models defined in ISO 14649. 134
Figure 4.27 An example of proto-persona for a user of risk assessment applications. 135
Figure 4.28 The user model for users of kinematic data exchange. 137
Figure 4.29 A vision proposal for a risk assessment system. 139
Figure 4.30 A vision proposal for a digital catalogue exchange system. 140
Figure 4.31 A vision proposal for kinematic data exchange. (“KIBOS” denotes KTH Implementation Based On STEP) ........................................ 140
Figure 4.32 A high-level reference model specified in ISO 14649 (author reproduction based on ISO, 2003b) ........................................ 141
Figure 4.33 Requirements of risk assessment application ...................... 147
Figure 4.34 Requirements of kinematic data exchange .......................... 148
Figure 4.35 An activity model related with machining operations (author reproduction based on ISO 14649 STEP-NC) ............................... 149
Figure 4.36 A reference model related with machining operations (author reproduction based on ISO 14649 STEP-NC) ................................. 150
Figure 4.37 A high-level conceptual model for data elements ............... 157
Figure 4.38 An excerpt of a data schema for STEP p105 ed2 (author reproduction based on ISO, 2014a) ...................................................... 159
Figure 4.39 A passive MVP pattern .................................................. 160
Figure 4.40 The classic concept of MVC ............................................. 161
Figure 4.41 Views and controllers are decoupled (Fowler, 2006) ............. 162
Figure 4.42 AM MVC (Sharan, 2015) .................................................. 162
Figure 4.43 MVP (Potel, 1996) ........................................................ 162
Figure 4.44 Passive MVP (Sharan, 2015) ............................................. 162
Figure 4.45 A flow chart to describe the process model ......................... 163
Figure 4.46 A class diagram to model the domain concepts .................... 164
Figure 4.47 A wireframe design for identifying a specific risk item ...... 166
Figure 4.48 A wireframe design for specifying risk items ..................... 166
Figure 4.49 An interaction diagram for an implementation integrating with existing CAX applications ......................................................... 168
Figure 4.50 An interaction diagram for an implementation as standalone application integrated with users’ existing workflow ................................. 169
Figure 4.51 A general interaction diagram based on MVC for both types of implementations ................................................................. 170
Figure 5.1 A unique process to implement STEP-based standards ........... 174
Figure 5.2 The modeling architecture using STEP Toolbox API ............... 175
Figure 5.3 Documented STEP Toolbox API by Javadoc ........................ 176
Figure 5.4 A flow chart for an integrated application to export a data set with full integration of kinematic data and geometric data .......... 178
Figure 5.5 A process design enhanced with STEP Toolbox API ............... 178
Figure 5.6 Process design of a kinematic data translator for CATIA ...... 179

XXIII
Figure 5.7 Mapping of hierarchy. .................................................................181
Figure 5.8 Prototypes for catalogue data exchange. .................................181
Figure 6.1 Implementers develop applications based on data schemas to
make data sets accessible for application users. .................................185
Figure 6.2 “Paradoxical loop” of implementing information models
(author reproduction based on Kiviniemi, et al., 2008)......................190
Figure 6.3 Discipline models and aggregate models in different stage of a
product lifecycle (Van Berlo, et al., 2012). .........................................193

Table 2.1 Different requirements on information models by extensibility
and portability. ..........................................................................................54
Table 3.1 Specifications to facilitate semiotic aspects of information
models standardized in the framework of STEP...............................83
Table 3.2 An excerpt of a p21 (conforming to ISO 10303-21) data set. ....84
Table 3.3 An example of an ISO 10303 data schema (a part of the long-
form schema of STEP AP242). .............................................................88
Table 3.4 Principles usable for representing information requirements. 92
Table 3.5 Specifications facilitating semiotics of implementation models.
.................................................................................................................94
Table 4.1 Working reengineering on a use case to specify risk assessment.
..............................................................................................................151
Table 4.2 Design framework of risk assessment...................................155
Table 4.3 Design framework of kinematic data exchange. .................156
Chapter 1
Introduction

The greatest challenge to any thinker is stating the problem in a way that will allow a solution.
- Bertrand Russell

Information technology (IT) has been serving engineers for decades in the field of product realization. A central artifact for product realization is product data\(^1\) stored, communicated, and processed in the form of models. In this respect, human-model interaction becomes a dominant activity of the product realization process. It is in step with the fashion of human beings to live in the Information Age: “Man the food-gatherer reappears incongruously as information-gatherer. In this role, electronic man is no less a nomad than his paleolithic ancestors” (MeLuhan, 1964).

When engineers are benefited from computerized information systems\(^2\), effective use of the product data becomes problematic. It is usual that the computerized information systems, instead of engineers, take control of how the product data can be used. A tool-driven manner\(^3\) (more details in Section 2.1.1) illustrates this phenomenon where the computerized information systems limit how users can interact upon product data.

To reuse and integrate product data, the core activity is system integration (Section 1.1.1) which leverages standardized information models\(^4\) (Section 2.2) as the major facilitator. The information models governed by

---

\(^1\) Product data: “A representation of information about a product in a formal manner suitable for communication, interpretation, or processing by human beings or by computers” (ISO, 1994), including data for lifecycle management, manufacturing, design, etc.

\(^2\) Computerized information system: A “human-machine system that provides information to support the operational, managerial, analytic, and decision-making functions” based on the use of computers (Krogstie, 2012; Falkenberg, et al., 1998).

\(^3\) Tool-driven manner: A problematic way to complete engineering goals greatly depending on tools in use with limited functions and content (Section 2.1.1).

\(^4\) Information model: “A formal model of a bounded set of facts, concepts or instructions to meet a specified requirement” (ISO, 1994).
modeling architectures could greatly impact implementation performance of system integration. Whoever using the information models will face the nature of its architecture—sophisticated semantic representations in an interdisciplinary context. It brings both troubles and opportunities. This thesis analyzes deep-down logic behind troubles of implementers and puts forth strategies for efficient system integration taking advantages of information models.

1.1 Research scope

As an interdisciplinary task, system integration based on information models is a difficult but crucial challenge for both software engineers and production engineers. This section explains the research scope with main concepts where the challenge need to be resolved. Two major concepts form the research scope, which are also indicated in the thesis title, i.e. model driven system integration and architecting activities.

1.1.1 Model driven system integration

System integration is the activity to meet the challenge of interoperability¹ (Section 2.1), as a process to progressively assemble “system components into the whole system” (ISO/IEC/IEEE, 2010). The components may be existing systems, envisioned systems or legacy systems. Data output by diverse systems may be fundamentally distinct and create obstacles between the systems. System integration is intended to fix these obstacles to satisfy requirements that cannot be met by these individual systems. It is the key path toward enterprise integration enabling connection between functional entities (i.e. devices, applications and people) beyond organizational boundaries and technical boundaries (Sherif, 2009).

To implement system integration, a type of standardized contractors (also known as protocols) should be identified (Figure 1.1). For CAx (Computer-Aided technology), product data is usually formalized as information models. Particularly, standardized generic information models (Section 2.2) are useful as the contractors to communicate product data. The major

¹ Interoperability: “The ability of two or more systems or components to exchange information and to use the information that has been exchanged” (ISO/IEC/IEEE, 2010).
reference of information models chosen in this thesis is the widely accepted standard ISO 10303 STEP (STandard for the Exchange of Product data).

![Diagram of System A, System B, System C, System D, Contractor](image)

**Figure 1.1** A standardized contractor in system integration.

Although the illustration looks very simple, a standardized generic information model is not the “one-stop” solution for system integration. Making an information model usable for all the systems could be a daunting task. Generally speaking, software implementation has “no silver bullet” (Brooks, 1987) because software engineers often dealt with requirements from other disciplines. The interdisciplinary nature is magnified when leveraging standardized information models.

To relatively ease the task, formal information models provide established modeling architectures: Schemas and methodologies (for representation, interpretation and processing) are specified as a package (more details in Section 2.2). Nevertheless, it is still not easy for implementers to learn and use such kinds of modeling architectures with domain-specific sign systems.

To support implementers’ work for system integration based on information models, this study proposes a solution named model driven system integration. It is based on the model driven approach (more details in Section 2.3), i.e. an approach using information models to integrate product data from different sources, to create a productive environment and to enhance business competitiveness. It implies that the information models act as a driver to exploit the potential of information technology for product realization.

As illustrated (Figure 1.2), the practices of model driven system integration involve at least three subject areas. Between the subject areas, architectures (Section 2.4) help to make use of knowledge from different subject areas and to manage complexity due to the interdisciplinary nature. The involved subject areas and architectures can be defined as following:
Model driven system integration: An interdisciplinary topic.

- **Software engineering:** “An engineering discipline that is concerned with all aspects of software production”. (Sommerville, 2011)
- **Information modeling:** Theories, approaches, tools and processes of development and use of information models (Section 2.2)\(^1\).
- **Product realization:** An engineering discipline studying all activities and functions (planning, controlling, operation and organization) directly contributing to making of goods\(^2\).
- **Integration architecture:** Structure, relations and rationales of system components for system integration (Section 2.1), decided at an architectural level.
- **Information architecture:** “(Human-centred) structure of an information space and the semantics for accessing required task objects, system objects and other information” (ISO/IEC, 2010, Section 2.5).

---

\(^1\) This definition is based on the definition of “modelling” by Knuuttila (2011), i.e. “construction and use of models”. Note that von Euler-Chelpin (2008) has also defined information modeling with the focus only on construction: “The activity of identifying, relating, and structuring the information types that need to be managed into an information model.”

\(^2\) This definition reuses the definition of “manufacturing”, in a broad sense, by CIRP (2004): “The entirety of interrelated economic, technological and organizational measures directly connected with the processing/machining of materials, i.e. all functions and activities directly contributing to the making of goods.”
• Modeling architecture: An architecture to specify development and use of an information model.
• Model driven system integration: An integration practice to facilitate interoperability between disparate systems based on the model driven approach (Section 2.3).

As illustrated, the model driven system integration builds a connection between modeling architectures and software engineering. This study is to change the way of development (Chapter 3) and use (Chapter 4) of information models, i.e. the modeling architectures, for the purpose of improve implementation performance of system integration.

1.1.2 Architecting

Architectures typically aim at managing complexity of interdisciplinary engineering tasks and communicating design ideas during system development. The architectures in this thesis indicate the interdisciplinary nature of implementing model driven system integration and the concern for human beings who are stakeholders of the implementation.

As suggested in Figure 1.2, architecting is the activity to resolve the interdisciplinary complexity with knowledge of many domains involved. An architecture (Section 2.4) is an engineering artifact to manage the application of knowledge to build a system (ISO/IEC/IEEE, 2011b). The three overlaps between each pair of circles in Figure 1.2 indicate architectures due to cross-domain integration, which are relevant for model driven system integration, i.e.:

• The integration architecture explores the best way to integrate computerized information systems for product realization.
• The information architecture utilizes and implements information models in software systems.
• The modeling architecture supports use of information models for product realization.

The three architectures are created to resolve complexity from two corresponding subject areas, and also strongly influenced by the third subject area. That is, the integration architecture heavily relies on utilization of information models; the information architecture should reflect requirements from production engineering; the modeling architecture is a facilitator for software implementation.
This thesis uses architecting also to express concern for human beings. Any system has many concerned stakeholders, e.g. end-users, developers, maintainers, customers. Development and use of a system implies application of domain knowledge of different stakeholders in different perspectives. Few people can understand all the relevant domains. Therefore, the architectures should address concerns of different stakeholders and integrate knowledge from different domains.

The architectures mentioned above are practically intermediate artifacts to support the work of different stakeholders whose final goals are applications to satisfy application users. How the stakeholders including application users work with the architectures and the final applications is illustrated in Figure 1.3. Indeed, application users are central stakeholders and applications are central artifacts to be produced. The user needs should be satisfied by the produced applications. However, all the architectures (in the middle of Figure 1.3) should serve the model developers and implementers (including application designers and constructors).

![Figure 1.3 Architectures and stakeholders in a context of the model driven system integration.](image)

Among the stakeholders, this study pays special attention to the implementers (application designers and application constructors) since they are direct users of information models. The mission of this study is to clarify their needs in relation to the modeling architectures and to reflect their needs in model driven system integration to enhance implementation performance: Firstly, to explore what is the best modeling architecture for the
implementers (Chapter 3); secondly, to clarify how implementers create integration architectures and information architectures utilizing a modeling architecture (Chapter 4).

1.2 A new modeling architecture
This section is a brief introduction of Chapter 3. The first part (Section 1.2.1) introduces necessary background knowledge, a challenge of implementation efficiency and a current incomplete focus of application contexts\(^1\) as the root cause of the challenge. The second part (Section 1.2.2) introduces how to resolve the challenge by dealing with the root cause, i.e. taking implementers into account when developing modeling architectures.

1.2.1 Pragmatic concern for implementers
For applying information models in system integration, implementers suffer from evident inefficiency (Section 3.2 and Paper D), which hinders wide practical use of many good information standards. Before proposing any solution to this problem, this study begins with locating the root cause, just as Joe Biden stated: “If you don’t understand what the cause is, it’s virtually impossible to come up with a solution”.

Currently, standardized generic information models can achieve semantic interoperability\(^2\) very well for computer interpretation, e.g. with the widely accepted ISO 10303. The achievement is based on computer interpretable data sets\(^3\) governed by semantically comprehensive data schemas\(^4\) (Figure 1.4).

![Figure 1.4 Information models to facilitate semantic interoperability. Both data schemas and data sets are information models.](image)

---

\(^1\) Application context: An environment where product data is used in a specific application (more details in Section 2.2.1).

\(^2\) Semantic interoperability: The ease to interpret meanings of exchanged messages by systems in an interoperation scenario (more details in Section 2.1.3).

\(^3\) Data set: An information model instantiating what defined in a data schema (more details in Section 2.2).

\(^4\) Data schema: An information model to represent types of information in a particular context (more details in Section 2.2).
However, successful computer interpretation of an information model cannot guarantee successful use of it (more details in Section 2.1.4). Human interpretation is inevitably involved because implementers should use data schemas and related data interfaces. Thus, the modeling architectures are expected to support implementers as well (Figure 1.5). This support is currently insufficient in popular standardized information models (e.g. ISO 10303), leading to inefficiency of implementation. This inefficiency could affect quality of an information model because use of any information model begins with implementation.

![Figure 1.5 Implementers interpret a part of modeling architecture to develop applications conforming to the data schemas.](image)

The insufficient support could be attributed to application contexts. Typically, the application contexts are used by implementers to design and develop application software. The application contexts are also used by model developers as an initial step (more details in Section 2.2.1). Nevertheless, when model developers only see the application contexts, primary users (i.e. implementers) of model developers’ work (i.e. data schemas) are neglected (more details in Section 3.2). For instance, implementation activities are not included in STEP Application Activity Models (AAMs) which represent the application contexts (more details in Section 2.2.2).

For standardization, translation from the application contexts to the data schemas should be formalized (more details in Section 2.2.1 and 2.2.2), which has negative effects: The focus drives complicated mapping mechanisms and complicated hierarchical modeling structures (more details in
Section 3.2). These complications may be understandable for model developers, but is uneasy to handle for normal implementers.

1.2.2 Development of a new modeling architecture

Hence, changes should be made for initializing modeling architectures based on application contexts. This study proposes implementation contexts\(^1\) (Section 3.3) as a new basis for modeling architectures. Section 3.3 describes what this new concept is and how it is used in the development of information models. The implementation context implies a necessary goal for a modeling architecture, i.e. to boost implementation performance in terms of efficiency\(^2\) and effectiveness\(^3\). On the other hand, an application context has a direct impact on applications and implementers and, therefore, is certainly a part of an implantation context. The inclusion of application contexts suggests that boosting portability of an information model shall not jeopardize its extensibility.

Enabling high extensibility and high portability at the same time, Kjellberg, et al. (2009) proposed a two-layer modeling approach composed of a generic stable core model and a specialized ontology. This study concretizes the idea of the two-layer approach as the concept of the implementation model\(^4\) (Section 3.4). An implementation model encapsulates and compensates data schemas to satisfy requirements in an implementation context. As a new layer in the modeling architectures, the implementation model is a usable abstraction of useful information for implementers. The abstraction means simplifying the complex modeling architecture, hiding what is unnecessary for implementation, and detailing pragmatic support.

With the proposed concepts, i.e. implementation contexts and implementation models, the first part of this study can be summarized as 1) using implementation contexts to accurately understand expectation of implementers on modeling architectures during integration implementation

---

\(^1\) Implementation context: An environment where implementers use an information model to implement an application (more details in Section 3.3).

\(^2\) Efficiency: “Resources expended in relation to the accuracy and completeness with which users achieve goals” (ISO, 2010b).

\(^3\) Effectiveness: “Accuracy and completeness with which users achieve specified goals” (ISO, 2010b).

\(^4\) Implementation model: An information model that encapsulates one or more data schemas, with additional supportive functions for data set manipulation, to satisfy information requirements in an implementation context (more details in Section 3.4).
(Section 3.3), 2) developing a methodology to specify implementation models to satisfy information requirements based on implementation contexts (Section 3.4) and 3) enhancing modeling architecture to facilitate development and use of implementation models in system integration (Section 3.5).

Based on the previous introduction, a research question and a corresponding hypothesis highlight the current challenges and the possible resolutions tested by this study:

**RESEARCH QUESTION I**
How can system integration be enhanced with information models?

**HYPOTHESIS I**
Introducing implementation contexts and implementation models in a modeling architecture improves efficiency of system integration.

Improvement of efficiency in implementing system integration is the core motivator of this research. For this purpose, an implementation context is firstly defined as an environment where an information model is implemented (Figure 1.6). Efficiency in an implementation context, rather than an application context, determines usefulness of an information model. Thereby, a modeling architecture should be developed to satisfy the implementation context where it is used.

The introduction of implementation contexts suggests a missing link in the development of the existing standardized generic information models. Hence, to interface the existing ones, an implementation model (Figure 1.6) is defined as an implementation-ready portable information model. For implementers, an implementation model acts as a bridge between their programs and data sets.
A typical use scenario is when an application software should conform to one or more standardized generic information models, possibly tailored. Encapsulating these information models, an implementation model can be realized as an API (Application Programming Interface) directly applicable for constructing such application software. An API is the most efficient and understandable way for implementers. To produce the API, the goal of the proposed modeling architecture will be as guidance to develop and to use an implementation model in a designated implementation context.

To develop and use the API, practical case studies are described in the appended publications A, B, C and E which propose implementation models for different purposes at suitable levels of abstraction. They are also validations of the modeling architecture in various instances of implementation contexts. As a summary paper, the appended paper D describes a Java implementation of the modeling architecture for efficient model driven system integration. The implementation in paper D is a basis for demonstrations in other publications (Figure 1.7). Paper D also makes an architectural comparison between the new solution and the original.
Within the publications, it can be observed that implementation models are more contextual, with less semantic details and more pragmatic details. Pragmatically, more implementation aspects in an implementation context make the implementation models less complicated and easier to use. The implementation models realized as APIs are constituted by components that are contextual for different system integration domains, e.g. kinematics and classification (Figure 1.7).

This contextualization does not jeopardize the reusability of the information models. The contextual components are harmonized to be reusable among applications in the broad application context designated by STEP. The harmonization has been validated in cases of kinematic modeling in paper A and E and also in cases of classification modeling in paper B and C (Figure 1.7). The paper A on kinematic errors is an extension to general kinematic information representation which is formulated in the paper E. The paper B creates a mechanism for cutting tool classification modeling. Then, the paper C describes a scenario to reuse the classification mechanism for catalogue information communication. In addition, all these use cases are harmonized with the product assembly modeling support as a part of the Paper D.

In conclusion, the first part (Chapter 3) of this study contributes to improving implementation efficiency with a new modeling architecture. To begin with, the causes of implementation efficiency issues should be discovered for system integration based on information models: 1) Development of data schemas is initialized by application contexts not including implementers; 2) in an application context, a large scope and universal acceptance at a semantic level require a complicated modeling mechanism;
3) changeable application contexts require highly extensible modeling architectures that is development-oriented but not friendly for implementers. Clearly, the use of application contexts is a weak link of conventional modeling architectures to support implementation. Based on implementation contexts to compensate the application contexts, a new modeling architecture can lead to information models satisfying needs of implementers. This solution forms the basis of implementing model driven system integration.

1.3 An integrated implementation process

Successfully implementing model driven system integration is not just about programming with information models as enabling infrastructures. Traditionally for implementing system integration, standards with different focuses are treated as technical constraints. A portable solution to use information models is not always available to be directly fed to programming. Programmers are faced with manual interpretation (Section 2.2.2) of sophisticated data schemas that are usually designed for computer interpretation. To resolve this situation, this study aims at guiding use (Section 1.3.2) of information models based on conventional software engineering lifecycle. It requires exploiting potential (Section 1.3.1) of information models in integration architectures and information architectures for system integration.

1.3.1 Two types of unused potential of information models

One type of potential of information models (e.g. ISO 10303) is embodied in highly comprehensive semantic representation for large scope, i.e. rich semantics in practice. The rich semantics leads to complex modeling architectures which have three results for different stakeholders (Figure 1.8): 1) It facilitates qualified model development and 2) formal computer interpretation; 3) it also result in heavy workloads for the direct end-users, the implementers. For implementers, they need smartly use, not suffer from, the complexity in software engineering activities, i.e. implementation.
This study claims that the use of an information model can be extended beyond what it is designed for, i.e. as a technical constraint (Section 4.1.1) to facilitate semantic interoperability. An information model has the potential of a reliable knowledge base underpinning an implementation process and facilitating pragmatic interoperability. Semantic comprehensiveness of an information model could help software design by saving effort at specifying domain specific functionalities and content. Thus, software engineers can focus on designing a system to present the functionalities and content in a user-interpretable way, rather than struggling with domain-specific semantics.

The other type of potential of information models is a similarity in procedure between information model mapping and software implementation. The procedural similarity is illustrated in Figure 1.9. Intended use of the information models, as a model mapping process, is similar to an implementation process integrating concern about HCD (Human-Centered Design) where research of contexts is highly valued for system design. (See more about HCD in an engineering context in the appended paper F.) Both processes transfer the same set of domain knowledge from contexts to deliverables through documented artifacts.

The procedural similarity can be utilized at different stages of implementation if a similarity in abstraction levels is taken care of. This potential will be discussed in Section 4.1.3 in depth, based on a context of general product realization. The detailed integration (Section 4.4) will be based on discussion of general stages (Section 4.2) and modeling activities (Section 4.3) of software implementation.
1.3.2 Make use of the potential in process integration

This integrated process is certainly a new way to perform implementation in an interdisciplinary context of model driven system integration. In Section 4.5 through Section 4.9, five stages with eleven activities are introduced to formulate the interdisciplinary process. Figure 1.10 illustrates the five stages of the new process in its diagonal, which displays the stages’ relations to the implementation activities and the information mapping.

Note that human-model interaction and communication is a significant feature in the application context of engineering activities (Hedlind, 2013). In this context, system integration is frequently built upon systems characterized by intensive human-computer interactions. The paradigm of
HCD is adopted in this study to address design concern for interactive systems\(^1\) (Section 4.3.1). In particular, presentation of content should be architected to ensure best human interpretability, findability and ease of navigation (more about information architectures in Section 2.5). Integration of all the subjects is a challenge for implementers who are often domain experts for system integration in practice (Section 4.3.2).

To support these implementers, the focus of the second research question is integration of the processes for model driven system integration. The processes to be integrated include the information model mapping, the conventional software engineering lifecycle models and the HCD. This integrated process should be model driven: Standardized information models, as a reliable resource for domain knowledge, are used to effectively underpin the software engineering activities.

**RESEARCH QUESTION II**
Where and how can information models be used to support system integration?

**HYPOTHESIS II**
Facilitated with existing modeling architectures and HCD (Human-Centered Design) principles, a software implementation process enables effective use of information models for system integration.

This study aims at a novel way to use the support from information models in system integration. Based on conventions in software engineering and HCD, stage-by-stage integration of the information models in implementation is resolved in Chapter 4. With this integration, implementers actively use information models in the suitable stages of the conventional processes, e.g. requirement engineering, architecture design and interaction design.

---

\(^1\) Interactive system: “Combination of hardware, software and/or services that receives input from, and communicates output to, users” (ISO, 2010b).
1.4 Main contributions

1.4.1 Identification of the root cause of challenges

The challenge lies in a mismatch between existing modeling architectures and the implementer needs. Implementation is the central activity of using the information models and is supported by modeling architectures. However, when modeling architectures are defined, needs of implementation are not captured. In particular, the data schemas are not sufficient to support implementation practices, and the designated ways to use an information model cannot support implementation efficiently.

Where does this problem originate from? Modeling architectures are mainly driven by how well a data schema can be developed (more details in Section 2.2), rather than by how well a data schema can be used. Standardized information models, e.g. ISO 10303 STEP, have prepared rich semantics in a broad scope for applications. Reasonably, model developers design a relatively complex modeling architecture to facilitate comprehensive standardization of rich semantics in a broad scope and an extensible fashion.

In this respect, benefits easily become weakness. This complexity could easily hinder pragmatic use of an information model, resulting in low portability and low implementability (more details in Section 3.2). For instance, several generations of the ISO 10303 modeling architectures have adopted more and more sophisticated multiple-layer structures to support development of comprehensive data schemas. The structures strive to ensure validity, completeness and extensibility of semantic representation in a data schema. However, the complex structures result in implementation difficulties such as instantiation inconsistency (von Euler-Chelpin, 2008). Without proper guidance (von Euler-Chelpin, 2008; Kjellberg, 2009), these structures could be a hurdle for implementation.

Figure 1.11 exemplifies a traditional modeling architecture of ISO 10303. As illustrated, model developers follow the upper process (development of a data schema) to specify a qualified data schema by creating a series of model artifacts. To use the data schema during a specific implementation project, a segment of the data schema will be located by implementers through the lower process (use of a data schema) in Figure 1.11.
This lower process could demand implementers to process plenty of documents (e.g. Chapter 2 of Paper D) to understand e.g. relevant content specified in a modeling architecture.

![Diagram]

Figure 1.11 The conventional modeling architecture of ISO 10303 STEP, based on textual description by Anderl and Wasmer (1997).

This modeling architecture (Figure 1.11) could cause technical problems and business problems. The technical difficulty is easily observed when applying such a modeling architecture in implementation. The modeling architecture leads to ambiguity of task allocation between roles of implementers. Practically, it is better if application designers perform the model mapping as exemplified in Figure 1.11 because they know the contexts and the requirements better. However, this is not intuitive (more details in Section 3.2) because, as a routine to use any standard, programmers should interpret and apply the standards as constraints in programming.

This mismatch between the modeling architecture and implementation needs could result in business issues on implementation. Firstly, it makes existing inadequate interoperability for product realization worse, by hindering efficient use of standardized information models. Secondly, the use of information models cannot effectively guarantee interoperability, in that conformity is totally determined by implementers who likely lack experiences with the modeling architectures; it can lead to incompatibility when implementations are based on incomplete assumptions. Thirdly, it is
a waste of resources that not many engineers can reap great benefits of the standardized information models as a comprehensive knowledge base.

1.4.2 Solutions

To tackle this situation, an information model can be either enhanced (research question I), or be used in an alternative way (research question II). The solutions aim at resolving the challenges by redesigning modeling architectures (Chapter 3) and the process (Chapter 4) to use the information models in system integration. Two strategies are proposed to take on the challenges for two aspects of modeling, development (Chapter 3) and use (Chapter 4). That is, developing information models should take care of its designated use scenarios, namely the implementation practices within the software lifecycle (Chapter 3). Use of information models should be aligned with software engineering activities from which integration architectures and information architectures are core deliveries (Chapter 4).

The two strategies form the backbone of this thesis: 1) Chapter 3 proposes a modeling architecture to guide a new way to develop an implementation-friendly information model. In the modeling architecture, concepts of implementation contexts and implementation models are introduced to satisfy the needs of implementers. 2) Chapter 4 designs a process to implement system integration that can use information models in an effective fashion. The purposes of the process include exploiting potential of existing standardized information models and speeding up and simplifying the implementation activities.

In addition to those two major chapters, the rest of the thesis fills in the background, validation and conclusions (Figure 1.12). This chapter briefly introduces research questions that are answered by the two chapters, in relation to hypotheses that shall be verified. Chapter 2 introduces the scope of study and reviews relevant topics. Chapter 5 describes several case studies that implement and validate the methods developed in Chapter 3 and Chapter 4. Chapter 6 concludes this study, with discussion of contributions, limitations and suggestions of future work.
1.4.3 New concepts

This study creates two new concepts to support development of modeling architectures. At first, this study defines an implementation context (more details in Section 3.3) as a primary environment where an information model is used. The implementation context plays an important role to understand what an information model is needed. With an implementation context, a modeling architecture provides formal representations of information suitable for implementing applications using an information model. On the other hand, an application context of an information model only lets model developers know representation suitable for application use.

This study also proposes an implementation model (more details in Section 3.4), interfacing data schemas, to facilitate efficient software construction based on the implementation context. The implementation context guides development of the implementation models as a new layer. A modeling architecture is developed to guide how to use the implementation models for model driven system integration.
The most important inter-related components in an implementation context are illustrated in Figure 1.13. An application operates on the information models and is interacted with by users in a context of use. If necessary, an application should be able to pool the information models with data from different domains to perform different tasks. Implementers perceive all the interactive needs as a context of use and, accordingly, find related information models to develop the application. Information models will be used by users, applications, and implementers at different levels of abstraction. That is, users possess data sets, applications operate on data schemas to process data sets, and implementers use implementation models to interface the data schemas. Note that these roles may overlap on particular persons in an implementation context (Section 3.3). End-user programmers are typical examples of this scenario, where an implementer of an application is also an end user of the application.

Figure 1.13 An application as an intermediate between stakeholders and information models.
Chapter 2
Frame of reference

Art and science have their meeting point in method.
Edward Bulwer-Lytton

Model driven system integration for product realization is a highly interdisciplinary topic (Section 1.1.1). Based on knowledge in different disciplines, this chapter sets the stage for this study. Interoperability (Section 2.1) is the most fundamental quality requirement of system integration. Standardized information models (Section 2.2) facilitate interoperability in system integration for product realization. Besides, the standardized generic information models can be treated as a reliable knowledge base for function design and content design in system integration implementation. In general, models are important infrastructures to maximize usability\(^1\) and reusability of information in engineering activities (Section 2.3). Implementation of system integration requires plenty of decisions about high-level structural designs; this leads to discussion about architecting activities (Section 2.4). Engineering applications are usually HCI (Human-Computer Interaction)-intensive and content-rich. This fact demands interaction design (IxD) at an architectural level adopting principles of information architecture (IA, Section 2.5).

2.1 Interoperability

This section will begin with description of the challenge (Section 2.1.1) that is represented by a lack of interoperability. It is followed by a discussion of previous frameworks to define interoperability as a quality requirement (Section 2.1.2). The discussed frameworks are noticeably originated from semiotics (Section 2.1.3). In the end, the pragmatic aspect is stressed as the focus of this study (Section 2.1.4).

\(^1\) Usability: “Extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”. (ISO, 1998b)
2.1.1 Interoperation as a challenge

The landscape of manufacturing industry has been significantly changed by Information Technology (IT). Competitiveness of IT enterprises is expressed by heterogeneity in technical infrastructures, operational practices and business strategies. The heterogeneity is particularly reflected in Computer aided technology (CAx) that has been changing the landscape of engineering activities in manufacturing industry. The industry demands CAx vendors to introduce more and more CAx functionalities for different business requirements based on different product lifecycle stages. However, no single IT product can satisfy all the requirements. Product realization heavily relies on interoperation of multiple types of applications. In addition, engineering activities frequently require experts working in a distributed manner (Wang, et al., 2002). Hence, for the interoperation, effective information sharing and exchange is needed.

Nonetheless, the heterogeneity could easily result in isolated systems with incompatible data. Moreover, continuous changes everywhere in industry increase discomfort of cross- and intra-enterprise communication and collaboration. Engineers are frequently challenged by unresolved interoperation issues with existing systems (ex-post compatibility) and new systems (ex-ante compatibility, David and Greenstein, 1990). This challenge was pointed out by Shannon (1948) in the dawn of the Information Age: “The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have meaning”.

Especially, what is communicated is with product data which can be created, processed and stored in a fragmented manner by disparate applications. The fragmented manner is exhibited by mismatches in syntax, semantics, or pragmatics (more details in Section 2.1.3). Fixing these mismatches by different strategies is the most important work to enhance interoperability.

Interoperation based on product data is required for both communicating existing systems and delivering new CAx functions to support professional engineering tasks. For such professional activities, implementing any functionality means dealing with rich content, e.g. models of geometry, properties and features. When content is available in existing systems, it is
not value-adding either to recreate a data schema from scratch or to ask users to recreate data sets from scratch. Hence, properly reusing those existing content is a key for efficient implementation. For instance, (Figure 2.1) implementers should consider to reuse the existing data schema when integrating a new function (e.g. kinematic modeling).

Figure 2.1 Reuse of existing data by implementers and users.

The above case particularly stresses that reuse of existing data facilitates efficiency of implementation. Implementers should concentrate on reuse of data schemas to enable application users to reuse data sets. Implementers have much work to do for successful reuse (more details about how to implement a data schema in Section 2.2.2). Implementation could be troublesome if there is no clear guidance toward the implementers. Hence, the implementers, as data schemas users, demands sufficient support to use the data schemas. Why and how the existing support is insufficient will be discussed in Section 1.2 (in brief) and Section 3.2.

Failing to reuse product data could result in a tool-driven manner which is noticeable in practical engineering activities. The *tool-driven* manner is a problematic way to complete engineering goals greatly depending on available tools rather than engineering specifications. It results
in limited choices of personnel, limited choice of tasks, limited ways to perform tasks and limited information at hand. These are all determined by ease to use necessary product data by applications suitable for the desired goals. These barriers to reuse the product data force engineers to settle for what are provided by proprietary tools. Otherwise, engineers have to manually process or re-type the data, when faced with non-reusable data sets or non-usable interfaces. Richard Stallman vividly illustrated this manner in the scope of application software several times in other contexts (TED, 2014, Figure 2.2).

2.1.2 How to understand interoperability
These challenges make interoperability a critical quality requirement when developing computerized information systems. The interoperability is “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” (ISO/IEC/IEEE, 2010). It is also a critical quality required to develop SoS (System of Systems, DoD, 2008). It relies on complicated integration solution based on a group of involved components and structures, e.g. interfaces, communicating entities, contexts, data, services and standards. This is a quality increasingly demanded in almost every domain related with IT, yet difficult to achieve appropriately (Brownsword, et al., 2004).
Previous researchers have defined different frameworks to depict typogrophy of this area, for instance, Martin (2005) put forth a three-level framework to classify interaction manners between systems:

- Unified systems: Interacting via same conceptual representation.
- Integrated systems: Interacting via agreed fixed representation.
- Interoperable systems: Interacting via dynamic interaction rules.

It is noticeable that integration and interoperation are similar in many contexts. Brownsword, et al. (2004) suggested that both terms can be used interchangeably, while integration may imply more tightly coupling, as what Martin (2005) pointed out in his framework.

There are a number of aspects that determine interoperability. Messages are the core elements to characterize interoperability at three levels (Pokraev, et al. 2007), which can be described with more concrete concepts:

- Syntactic interoperability: Formats are compatible.
- Semantic interoperability: Meanings of messages are shared.
- Pragmatic interoperability: Expectation of effects of messages are shared.

Based on this work with extension, Sharif (2009) developed a similar four-stage framework to describe system integration, with emphasis on inter-level dependency. The stated here were suggested to be implemented sequentially toward best performance of system integration:

- Interconnectivity: Telecommunication infrastructures are facilitated.
- Functional integration: Networks, protocols, formats and procedures are defined technically for integration.
- Semantic integration: Consistent semantics is achieved.
- Optimization & innovation: Integration enables improvement of technology or business processes.

### 2.1.3 Semiotic aspects of interoperability

Obviously, the frameworks to define interoperability are influenced by basic semiotic concepts. Scope of semiotics can be defined as a famous trichotomy, syntax (or syntactics), semantics and pragmatics. Morris (1938) coined this trichotomy and identified it as three branches of semiosis. The other division made by Morris (1938) was so-called pure semiotic and de-
scriptive semiotic. The two division systems form a structured view to describe the scope of semiotics. The trichotomy was considered to be inherited (Posner, 1985; Nöth, 1990) from the medieval trivium (grammar, dialectic and rhetoric, Greene, et al., 2012) and Peirce’s triadic model of the sign (representamen, object and interpretant, Nöth, 1990).

Although validity of this trichotomy has been questioned since its introduction (Lieb, 1971), the triad is still a useful tool to describe a “sign-situation” (Figure 2.3, Morris, 1939), e.g. representation, abstraction, modeling, etc. The triad includes three components of semiosis and three dimensions of semiosis, which lays an important early foundation for research on modeling. Many researchers on information systems have applied, integrated, and extended this triad into layers to discuss relevant concepts.

![Figure 2.3 “A sign-situation, or a process of semiosis” (drawing based on Morris, 1939).](image-url)
Dealing with conceptual modeling for requirement engineering, Lindland, et al. (1994) made a remarkable early contribution to adapt this tri-chotomy in the modeling domain of software engineering. In their framework, four basic sets were defined for possible statements in a conceptual model (Figure 2.4):

- **Language** is a set of all statements in a syntax. The syntax including an alphabet (a set of modeling construct) and a grammar (a set of composing rules).
- **Domain** is a set of all statements to solve a specific problem. It includes necessary knowledge for a specific context, instead of an entire context of a discipline.
- **Model** is a set of statements, including statements explicitly made and statements implicitly deducted according to the “language”.
- **Audience interpretation** is a set of statements in the “model” understood by audience, correctly and completely.

![Figure 2.4 Four cornerstones and three connections of the framework proposed by Lindland, et al. (1994)](image)

**Model, abstraction and level of abstraction**

To put it simply, a *model* is a collection of representations to describe an existing or a future system (Blanchard and Fabrycky, 1990). Minsky (1965) defined a *model* from the perspective of purposes, “To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A.” IEEE (1998a) defined a *model* as “a representation of a real-world process, device, or concept”, while the characteristic of the model to suppress certain aspects was emphasized by IEEE (1998b). This study adopts the definition of a *model* concluded by ISO/IEC/IEEE (2010), based on previous ones defined by ISO, IEEE and IEC, as “a semantically closed
abstraction of a system or a complete description of a system from a particular perspective.” In an information model, it is critical that represented concepts and their relationships should be used to specify content which matters most for a subject domain (Hackos, 2002). Of course, there should be rules, constraints and operations to enrich semantics (Lee, 1999). Specifically, this thesis uses the term “model” in three contexts:

1. An information model in the domain of product realization is a formal model to capture a bounded set of information to meet a specific requirement (ISO, 1994). By this definition, all CAx should be developed with information models to some extent. The model driven approach will be defined in this context as well.

2. A system design model in the domain of software engineering is a model to describe contexts, requirements, architectures and designs as deliveries of model based design activities in a software implementation process.

3. A data model as a component of a software architecture. It is an important part in the architecture pattern MVC (Model-View-Controller, more details in Section 4.8 and 4.9).

ISO/IEC/IEEE (2010) took a model as a kind of an abstraction and it also defined an abstraction as “a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information.” Abstraction as an action indicated a selection process (Ross, et al., 1975), i.e. to extract relevant information but to omit irrelevant details. In conclusion, as an entity, an abstraction denotes a model describing a system for a purpose; as a procedure, an abstraction uses levels of abstraction to construct “a view of an object at a specific level of detail.” (ISO/IEC/IEEE, 2010) At a higher level of abstraction, more important information is emphasized and typically less details are revealed.

In this framework of Lindland, et al. (1994), model developers were supposed to be concerned with appropriateness between a “model” set and the other three sets. These concerns were addressed by the three qualities (Figure 2.4):

- Syntactic quality: “How well the model corresponds to the language”.

30
• Semantic quality: “How well the model corresponds to the domain”.
• Pragmatic quality: “How well the model corresponds to its audience interpretation”.

For each quality, practical issues in terms of goals, means, validity, completeness and feasibility were represented by set theory notations. With this solution, Lindland et al. (1994) offered a systematic way to organize and utilize those principles for modeling in software engineering. Their framework aims at a deeper understanding of modeling qualities than previous attempts consisting of just unorganized quality properties. Until now, researchers on conceptual models or software architectures are still producing these “bread-and-butter collections”, e.g. annotated, appropriated, clean, formal, modifiable, traceable and verifiable.

Hofmann (2003) extended this use of Morris’ trichotomy to integration of model based computerized information systems in the domain of modeling and simulation. In his study, models were no longer statements used for conception, requirement analysis and architectural design. Instead, the models were used to manage “tremendous complexity” of reality. In other words, the models were regarded to be a substitute for designing or controlling state transition of a real system (Figure 2.5). In this illustration, real dynamics of a system $\beta^S$ was too complex to be directly understood and managed. A model was supposed to be developed with necessary abstraction, idealization and simplification during $\varphi$ (Figure 2.5). Model dynamics, $\beta^M$, described or predicted the system dynamics. After execution of model dynamics, a computerized information system could “retranslate” the new model state to a new system state, in $\psi$.

![Figure 2.5 Model based computerized information systems (reproduction based on Hofmann, 2003).](image-url)
Integration of model-based computerized information systems was all about harmonizing different models, so that content and functionalities in different systems could be shared and reused. Technics, syntax, semantics and pragmatics were identified as essential preconditions concerning model development for system integration:

- **Technical aspects** were realized by selecting formal protocols for networking and interfacing. Besides basic coupling functionality, a feasible protocol should fulfill needs in flexibility, maintainability and additional services (including authorization, time management, security, etc.)

- **Syntactic aspects** required “automatically processible data” in a formal modeling language. The modeling language could be challenged by 1) a lack of personal learning incentives, 2) a lack of expressiveness and 3) difficulties of understanding.

- **Semantic aspects** required “a predefined ontology” as a glossary to attribute meanings to syntax. Besides, semantics for model dynamics ($\beta^M$) was difficult to integrate only with a static standardized ontology. Hence, Standardized algorithms for elementary processes in the domain were also required.

- **Pragmatic aspects** required model dynamics ($\beta^M$) in a consistent manner after the integration. It led to standardized $\beta^M$. However, it was always difficult to harmonize context, broadness and resolution of models even with same syntax and semantics. A less compulsory solution is to standardize graphical user interfaces (GUI), although it is not realistic.

Influenced by these preconditions, Tolk (2004) proposed his framework to describe levels of conceptual interoperability, where data is considered as the centric attribute in these five levels of system interoperability (Tolk, 2003). The target of Tolk (2006) was also to maximize interoperability at a pragmatic level. The framework is called LCIM (Level of Conceptual Interoperability Model), of which the first version is composed of six levels:

- No connection at all.
- The technical level: Physical connectivity is established.
• The syntactical level: Standardized formats and protocols are established for data exchange.
• The semantic level: Common reference models are established for information (data + contexts) exchange.
• The pragmatic/dynamical level: An unambiguous form is established for knowledge (use and applicability of information) exchange.
• The conceptual level: A common view of the world (epistemology) is established for theory of domain knowledge (e.g. reference to its limits and validity) exchange.

2.1.4 Pragmatics: A focus of this study
Synthesizing the above categorization theories as well as insights gained from project experiences, this thesis selects three most important and feasible levels of interoperability: Syntax, semantics and pragmatics (Figure 2.6). A higher level indicates easier integration (Tolk and Muguira, 2003). The pragmatic level of interoperability is the uppermost need regarding information for implementation of interactive system in practice. Note that the pragmatic interoperability in this study focuses on a technical level rather than a business level (Asuncion and Van Sinderen, 2010). Namely, it requires mutual understanding on data intention (expected effects), data use (actual effects) and a context where communication happens (Liu, 2009). The modeling architecture (Chapter 3) reaches this understanding by employment of the implementation models based on the implementation contexts.
Figure 2.6 Three levels of interoperability concerning this study.

The levels of semantics and syntax are foundations for the pragmatic level. In general, consistent syntax and semantics can assure acceptable interoperability for many cases without human involved. As illustrated in the upper part of Figure 2.7, meanings (e.g. product information) in communicated data sets are interpretable by computers. Afterwards, the computers decide how to present the meanings to application users and what kinds of manipulation on the meanings can be done by the application users. To an extent, this is helpful for the application users, as computers are much better to store information massively, access information in a focused way, and to reason beyond human cognitive ability (Arp, et al., 2015).

However, until now, to make the semantic interoperability a reality, human interpretation is inevitable in that human implementers create all the functions to process the semantics. When the meaning in data sets is automatically presented to users, implementers have to find a way to manually interpret the meaning of the data schemas (the lower part of Figure 2.7). Taking implementers into account, performance of human interpretation of data schemas is represented by pragmatic interoperability.
This study uses the three-level framework in two ways. Firstly, in Chapter 3, a modeling architecture is proposed to facilitate pragmatic interoperability. A missing link in development of information models is identified and resolved in a context of model driven system integration. It corresponds to Hypothesis I. Secondly, Chapter 4 takes advantage of the high semantic interoperability of standardized information models to underpin an implementation process (Hypothesis II). This measure is critical for implementers because of the interdisciplinary nature of the implementation activities.
2.2 Standardized information models

The system integration requires qualified information models. Standardized protocols determine success in interoperability and integration (Martin, 2005). Architecting system integration means exploring a best way to navigate data through the protocols. There are numerical styles of the navigation: Shared database, file transferring, remote procedure invocation, message bus, etc. (Hohpe and Woolf, 2003) No matter which style is chosen, an information model has to be used to represent information to be exchanged. Hence, success of system integration depends on not only interoperability of applications, but also interoperability of information models.

In particular, important facilitators for system integration are standardized generic information models, of which specifications are known as generic information standards. A major capability of such standards is the interoperability for communicating meaningful information (Kazman, 2014). The standards are also known with a broad scope and wide applicability which can be expressed by the generic PPR (Product-Process-Resource) framework proposed by Nielsen (2003). Examples of the generic information standards are AP214 (ISO, 2014a) and AP242 (ISO, 2014b) which form the base of development, use and discussion of modeling architectures in this thesis. Extensive discussions about benefits of the generic information standards have been made by von Euler-Chelpin (2008) and Kjellberg et al. (2009). In brief, benefits of the standardized generic information standards include:

- Qualified semantic representation,
- Reliable methodologies for development and use,
- High extensibility.

This section will introduce key concepts for information modeling at first, e.g. application contexts, information requirements, data schemas and data sets. These concepts are critical for specifying an information model in a modeling architecture.

**Information model, data schema and data set**

An information model, as a central concept for model driven system integration, was defined as a “formal model of a bounded set of facts, concepts or instructions to meet a specific requirement.” (ISO, 1994) An information
model can be composed by an interrelated or integrated set of information models (Schenck and Wilson, 1994), particularly in forms of widely accepted information standards, e.g. ISO 10303. For instance, the data schema of ISO 10303-214, integrating several information models, is a widely used standardized generic information model. This study only focuses on the standardization effort when discussing information models in general, as the need of system integration.

There are many ways or criteria to categorize information models. This study is concerned about one way to categorize information models, according to levels of abstraction of represented information, into two categories, data schemas and data sets. A data schema is an information model to represent types of information in a particular context. This is a definition combining an old definition of “information model” by Schenck and Wilson (1994) and a more specific definition of “application schema” by ISO (1998a). With the definition of a data schema, a data set is an information model instantiating what defined in a data schema. A data schema formally specifies conventions in syntax and semantics to construct a data set in a computer interpretable way. According to the definitions of the information model and the model, both the data schema and the data set are supposed to represent a system to meet a specific requirement. The system described by a data schema has a larger scope and a higher level of abstraction than the one described by a data set.

2.2.1 Application contexts and information requirements
Design of an application shall begin with understanding the context where the system is used, so shall design of an information model. It is always easy to draw fascinating graphical interfaces, to invent tempting functions and to implement complicated algorithms. However, it is the context that justifies whether the developed systems are useful or not. Development projects without adequate analysis on contexts of use can end up with systems dissatisfying user needs (Bevan, 2013).

Software implementers took decades to realize that their deliveries should be human-centered, rather than programmer-centered (Grudin, 1990). Human-Centred Design (HCD, ISO, 2010b) starts from understanding contexts of use. Its importance was confirmed by famous IT system design methodologies such as scenario-based design (Rosson and Carroll, 2009) and goal-oriented requirement engineering (Van Lamsweerde,
The contexts help implementers to obtain complete, pertinent, traceable, readable, stakeholder-oriented, conflict-free requirements (Van Lamsweerde, 2001) to be implemented.

To understand the requirements of what should be represented, ISO 10303 devised the concept “application context” (ISO, 1994) which is useful for information modeling (development and use/implementation). Figure 2.8 illustrates how an application context is used in software implementation. The application context forms a basis for implementers to produce applications for application users. In an application context, there may be other components than just application users: Tasks, domain knowledge, technological environments, business environments, etc. This idea seems promising also for specification of information models that are obviously in a similar subject domain and a related problem space. As defined in Figure 2.9, an information model is developed based on a designated application context and is intended to be used by a group of processes (an application) situated in the application context. This pattern can be observed in widely used standards such as ISO 10303 STEP.

![Figure 2.8 Application context used in software implementation.](image)

![Figure 2.9 The application contexts defining information models.](image)
In the domain of software engineering, a requirement is a “statement which translates or expresses a need and its associated constraints and conditions” in a high-level form (ISO/IEC/IEEE, 2011a). This definition can be applied to a system as a whole to accomplish an objective, or to a software component of a system of which elements include hardware, firmware, people, information, services, etc. (INCOSE, 2010) The requirements represent needs of not only users or operators, but also other types of stakeholders, e.g. customers, regulatory authorities and maintenance men/women.

Software engineers specify requirements (namely, intended use of a system) based on an application context (ISO, 2003a) which is an environment where data is used in a specific application (a modified definition based on ISO, 1994). Toward common interactive systems, a similar and newer concept is a context of use, “users, tasks, equipment (hardware, software and materials) and the physical and social environments in which a product is used.” (ISO, 2010b) This definition used a larger scope with identified components that constituted a context in relation to users for the system-of-interest. A context can be treated as a function of the components. A system is normally designed to work in a range of context of use, i.e. several combinations of the contextual components. A complete description of a designated context is not realistic for any system, because a large number of contextual factors are essentially meaningless for a specific application. Moreover, a system in any stage of its lifecycle (e.g. development, deployment and production) may reshape the context for the system. During a lifecycle of a system, some components may be subject to changes but some may not. System designers need to decide a set of ranged aspects that are relevant for a system-of-interest and their changeability (ISO/IEC, 2014).

As it was proposed for, this study delimits the concept of the application context only applicable to information models and applications using the information models and that an application can be specialized by application software, “software that is specific to the solution of an application problem” (ISO/IEC, 2015). In requirement engineering, there should be many types of requirements elicited from a context and information models and applications should be based on different contexts and different consequential sets of requirements. Traditionally, an information model is treated as a technical
part of an application, so an information model can be defined by a subset of requirements necessary for developing an application. The subset is *information requirements* (Figure 2.10), i.e. a set of requirements in an application context that should be satisfied by a data schema. ISO (2016c) defined three types of application contexts that can be used to specify information requirements: 1) Exchange, 2) integration and sharing and 3) long term archiving and retrieval.

![Diagram](image.png)

**Figure 2.10** A traditional view of information requirements as a subset of requirements to be implemented in information models.

However, a lack of distinction of data sets and data schema makes it ambiguous based on this understanding of the relation between application contexts and information models. A data schema is developed to describe a way to represent information supporting an application context. A data set, (probably partially) instantiating a data schema, is operated by an application to perform some tasks in an application context. For system integration, the applications shall comply with a particular data schema to correctly interpret the corresponding instance data sets from other sources, in order to achieve interoperability. Thus, Figure 2.9 can be extended to Figure 2.11.

![Diagram](image.png)

**Figure 2.11** A data schema and a data set.
If only considering use of information models in implementation, a more detailed illustration can be obtained in Figure 2.12 combining Figure 2.8 and Figure 2.11. Some new concepts are also introduced: 1) Data sets are representations of product information used by engineers; 2) With the help of data sets, product information can be output and be shared with others, i.e. customers; 3) an application context may include other types of components, e.g. product information and customers; 4) the relation between application contexts and data schemas are omitted, since creation of data schemas is not relevant in implementation. 5) Information models, in the form of data schemas and data sets, are technical constraints for implementation. For instance, a product designer uses a CAD (Computer-Aided Design) system to produce a product design model and then hands the model to a process planner. In this case, the product designer is an application user, the process planner is a customer and the product design model is a form of product information represented in a computer-interpretable data set. Customers do not care about how well an application is developed as long as the product information is well captured for their own tasks. Therefore, the customer is an indirect user (ISO/IEC, 2014) who does not interact with the application directly but is affected by the application’s output indirectly.

Figure 2.12 Application context, the current basis for application development.
Meanwhile, as illustrated in Figure 2.11, application contexts are also used to develop data schemas. Thus, Figure 2.12 can be extended to Figure 2.13 with additional blue elements. Model developers are introduced in this new illustration to make use of application context and to create new information models. As illustrated, the major output by model developers is data schemas which will be used by implementers directly.

![Figure 2.13 Application context, the current basis for both application development and information model development.](image)

As stated in the beginning, methodologies to use application contexts in application development are mature in the domain of software engineering and HCI. It is time to look at how to use the application contexts in information model development, which has been described briefly in Section 1.2. A general development process is displayed in Figure 2.14. At first, activity models are used to describe application contexts. Developers elicit
information requirements from the activity models and create reference models to formally specify information requirements. In the end, according to the reference models, data schemas are developed to satisfy all the information requirements. This mapping process is a theoretical generalization of STEP modeling architecture which will be described in the following subsection.

![Diagram](image)

**Figure 2.14** Application context used for information model development.

### 2.2.2 The STEP modeling architecture

To discuss behaviors and structures of information models, this study need a reference widely-accepted in production engineering. For this purpose, this study uses a standard family, ISO 10303 STEP and related ones (e.g. ISO 13399, ISO 13584 PLib and ISO 14649 STEP-NC), as a main utility. Within the standards, almost all relevant semantics of product data have been standardized. Moreover, this standard family is known of a comprehensive modeling architecture formally specified and documented: It includes modeling languages, specification of application contexts, specification of information requirements, computer interpretable data schemas, development methods, implementation methods, validation methods, etc.

Modeling architectures are guidance of development and use of information models. For development, the traditional modeling architecture (ISO, 1994) is illustrated in Figure 2.15. It is a basis to derive other new modeling architectures (ISO, 2016c), e.g. an MIM (Module Interpreted Model)-based architecture constructing AP242. In a modeling architecture, the main goal is an exchangeable data set (in the bottom of Figure 2.15) conforming to an AP (Application Protocol). It instantiates a (part of) AIM (Application Interpreted Model) which integrates resource constructs to
satisfy information requirements represented by ARM (Application Reference Model). This mapping process is called “interpretation” which contextualizes resource constructs for particular information requirements. To represent information requirements, the ARMs use UoFs (Units of Functionality) or other modularization techniques to categorize conceptually interrelated application objects. In a well-documented AP, a set of AAMs (Application Activity Models), usually in a hierarchy formatted in IDEF0 (Integrated computer aided manufacturing DEFinition for Function modeling), describes how the UoFs or the application objects work in an application context. Moreover, syntax, in terms of modeling languages, should be defined, e.g. EXPRESS (ISO, 2004) for ARM, AIM, interfaces and conformance testing; IDEF0 for AAM; and p21 (ISO, 2016a) or p28 (ISO, 2007b) for data sets. In this way, a neat mapping process is defined from application contexts to data sets.

Figure 2.15 The traditional modeling architecture of ISO 10303 STEP.
This relatively complex structure of modeling architecture can be understood from a semiotic perspective, where semantics and syntax have been well defined. It is necessary in that engineering activities for product realization in a digital context relies on precise processing of tremendous amounts of information. In such a domain, it is often impossible for human beings to effectively manage and parse information entities (universals and particulars) without computer assistance (Arp, et al., 2015). Standardization of information models up to a semantic level is a critical strategy to facilitate consistent interpretation by computers. In a professional domain, standardization for interoperability at a semantic level or higher is difficult (Ouksel and Ahmed, 1999). The semantic quality of a model is “how well the model corresponds to the domain” and two important components are completeness (broadness plus granularity) and validity (Lindland, et al., 1994). High semantic interoperability demands a high level of formality (Ouksel and Sheth, 1999) to specify broadness, granularity, and validity. In STEP modeling architecture, AAM (for broadness), ARM (for granularity), and modeling languages (for validity) exhibit the formality.

Besides development of a data schema, the modeling architecture is also about use of a data schema by implementers. To help implementers, artifacts more than models are provided (Figure 2.16). In this illustration, general terms defined in Figure 2.14 are specialized in the STEP context. Conformance test tools and SDAI (Standard Data Access Interface, ISO, 1998a) are the additional for implementers to interact with data sets conforming to the data schemas (AIMs).

![Figure 2.16 Artifacts (rectangles) in STEP modeling architecture useful for implementers.](image)

Implementers’ final goals are 1) to locate useful parts (the smaller block named “Useful part of AIM” in Figure 2.17) out of data schemas
(AIMs) and 2) to construct application software conforming to these parts of data schemas. For the first goal, implementers should find out a generally useful part of AAMs which can describe a context of use where the envisioned software is situated. Then implementers should identify a suitable set of ARMs, i.e. a set of information requirements corresponding to content requirements; this identification cannot be fully based on the selected part of AAMs because in most APs the mapping between AAM and ARM is not formalized. In the end, a valid part of data schemas (AIMs) can be obtained based on the set of ARMs. For the second goal, implementers should learn to use SDAI and conformance test tools combined with the part of AIMs, in order to construct the application software. The SDAI processes data sets at a syntactic level; namely, semantic correctness of processes cannot be ensured. Hence, conformance test tools are necessary to guarantee semantic correctness, i.e. completeness and validity. Issues regarding this implementation procedure will be discussed in Section 3.2.

One example of the above-mentioned procedure to generate a useful part of AIM is discussed in paper A where basic kinematic data exchange applications have been implemented. A beginning point is a CAD plugin (for Siemens NX) to export/import kinematic information; it is also required to be seamlessly integrated with geometric information in a form of

Figure 2.17 Software development based on the STEP modeling architecture.
STEP AP214. In practice, the application occupies a small part of the information flows in the application context of AP214. All useful concepts for the kinematic modeling are collected in one AAM-data class and is used within one UoF (Kinematics, K1). This AAM-data class is referred to in two defined information flows: Concepts and product description, as outputs of two processes: Styling product and designing product, both of which are also introduced in ISO (2014a). In UoF K1, a list of application objects (ARM entities for unique concepts) is available, most of which form the needed segment of ARMs for content requirements of the application (e.g. kinematic joints, kinematic links and kinematic pairs). In the end, a useful part of AIM can be generated from associated mapping tables of these application objects.

Software construction (a late stage in software implementation) is introduced with a case in a part of Paper D, where use of SDAI and conformance test tools are discussed. SDAI is the most direct information models and the designated user interface for implementers. In the procedure of implementation (Figure 2.18), a useful part of AIM is actually an indirect resource used by implementers. The procedure begins with selection of APs potentially useful for the context of the developed software. A useful part of AIM is one result which should be manually obtained by implementers based on model artifacts (AAM, ARM, AIM, etc.) of APs.

Meanwhile, the AIM as a data schema can be compiled automatically by a SDAI toolkit (there is also a brief introduction of toolkits in paper D). Then, implementers can start programming supported by the compiled schemas and general operations provided by SDAI. A resultant program should be tested through data set validation against the compiled schemas.
Figure 2.18 A workflow to implement information model standards based on STEP modeling architecture.

Note that the information models should not affect effectiveness of implementation. No matter how broad a standard is defined, it is still possible that its designated application context cannot cover all the implementation needs. Especially in an interdisciplinary scenario of system integration, information models are sometimes used in a context that are not pre-defined. It is also possible that different protocols are used together, so that a holistic view of information can be presented to application users. For instance, to enable interoperable classification (paper B) and categorization (paper C), engineering applications need collaborative operations from several sources, i.e. manufacturing resource (e.g. cutting tools) models to define semantics of dictionaries or catalogues, process models to specify usage of the resources and product design models to describe the products to be realized (the left side of Figure 2.19). These kinds of tasks require more than one type of information models to be integrated and implemented. Meanwhile, the right side of Figure 2.19 illustrates user needs for multiple
viewers to process information holistically from different viewpoints. Each viewer represents a style to present data the information models convey.

![Diagram](image_url)

**Figure 2.19** Information models from different domains are used by applications focusing on applying multiple views of product data.

2.2.3 Extensibility and portability of STEP

This modeling architecture demonstrates the four major requirements of an information model: Extensibility, portability, interoperability and longevity (Al-Timimi and Mackrell, 1996). For the scenario of system integration, interoperability is why a standardized generic information model is needed and selected in the first place. Longevity of an information model depends on how accessible and reusable it is, which is all about the interoperability and portability. Thus, extensibility and portability become two most critical qualities that determine how valuable an information model is for users and how interoperable and long-lived it is eventually.

Al-Timimi and Mackrell (1996) defined the *extensibility* as an ability to “continue to take advantage of new and innovative techniques” emerging from continuous evolution and the *portability* as an ability to “move data among applications”. More generally, ISO/IEC/IEEE (2010) defined extensibility (i.e. “extendability”) as “the ease with which a system or component can be modified to increase its storage or functional capacity,” and portability as “the ease with which a system or component can be transferred from one hardware or software environment to another”. It is clear that extensibility is about internal structure of a system and that portability
is about external use of a system. Both qualities are useful for system integration, in that extensible information models can easily integrate information from different contexts and that portable information models can be easily reused in different applications.

In other words, a changing environment is the root cause of “extending” or “porting” an information model. An information model can take effect only when it works in a using environment. Application contexts and applications are two major constituent elements in this environment. It is the various application contexts and the various applications that constitute the changing environment. Any information model has a potential to be used among many other applications in a designated (or overlapped) application context (Xu, 2009). However, again, it is vague to understand the extensibility and the portability without distinction of data schemas and data sets when it comes to applications and application contexts (Figure 2.20).

An extensible information model should facilitate efficient data schema extension to meet new information requirements. The new requirements for system integration usually cause changes in processes and information flows, i.e. the application contexts and thus require an extension of data schemas (Figure 2.21). It is also possible to extend data sets, but how much a data set can be extended is delimited by the instantiated data schema, so this thesis only concerns extensibility as a preferred quality for data schemas. Besides, backward compatibility is also an issue for extending a data schema, i.e. the extended information model will still be effective with the applications complying with the previous version of the information model.
On the other hand, a portable information model should facilitate efficient reuse of information represented in a data set by a new application. The data set may be created or exported in another application complying with an instantiated data schema. Although porting the data set is most important for application users, implementers should also, in advance, port the instantiated data schema to the application to automate the reuse of the data set. It is the portability of an information model that determines the ease of making the new application conform to a data schema and of interpreting a data set (Figure 2.22). A simply porting process reuses an existing data set without any change in content and the porting process is executed during an implementation process. The extending process and the porting process can be easily distinguished by whether there is a change in the content of the information models.

Hence, these two abilities actually take care of different stages of the technical process of the information models, i.e. the development stage and the operation stage of an information model. The data schema and the data set are two results for the corresponding stages, respectively. Extension of content of the data schema is the major process to be evaluated for extensibility. Evaluating portability should evaluate the process performance of porting the data set and the instantiated data schema in different applica-
tions. There are different aspects to describe the performance when evaluating the abilities. Efficiency and effectiveness are important metrics to evaluate the extent of the two qualities. Efficiency is obviously a major focus to evaluate extensibility and portability, if high delivery accuracy of the extending process or the porting process is assumed.

2.2.4 High extensibility is preferable

High extensibility is a preferable quality for information models for many reasons. Model developers decide what quality an information model should possess. High extensibility is obviously prioritized since it is highly related with the ease to develop information models. Besides, information models are expected to support an application context as broad as possible to enable high readiness for system integration. An extensible model is hopefully easy to be reconfigured to meet new requirements in an extended application context.

There are plenty of implemented approaches to enable high extensibility in general. Agile modeling strategies to enable development of information models to meet changing requirements efficiently can be traced back to standardization of database management systems, with three schemas for presentation, logic and representation (Tsichritzis and Klug, 1978). Therein models can evolve independently of applications. The idea of decoupling contexts from data is an important step to enhance extensibility. According to this decoupling principle, an information model should be independent of the applications using it. In other words, what the model is used for is more of concern than how the model is used. Besides, an information model should be constructed inherently with reusable building blocks. The reusable building blocks are information representations at different levels of abstraction. STEP AP242 constructed by MIMs (Model Interpreted Model) is one example with a hierarchical set of building blocks. With this manner, different application contexts can be supported with different composing strategies (Figure 2.23). One building block may be constructed with other building blocks at a higher level of abstraction and a lower level of hierarchy. The dashed line in Figure 2.23 indicates that a data schema describes a part of an application context, in terms of information requirements. Last but not least, an application context defining an information model should be broad, coherent and semantically closed so
that possible instances (the data sets) have sufficient extensibility and that the interoperability of the using applications is not constrained by the data schema too much.

![Diagram of Reusable building block, Data schema, and Application context]

Figure 2.23 Reusable building blocks to compose an information model.

Just like any other types of systems, portability and extensibility are non-functional requirements stated informally and sometimes controversially (Mylopoulos, et al., 1999). The portability requires an information model to care about its accessibility and reusability by a specific application (with data sets) or a specific implementation practice (with data schemas). In other words, how to use and reuse the information models matters for portability. High portability requires efficient implementation of a data schema in an application so that valid data sets can be produced or interpreted by the application. This process is usually performed manually by implementers. Hence, the modeling architecture (probably in a complex hierarchy with building blocks at different levels of abstraction) should be either easy to comprehend by implementers, or hidden with a simplified implementation architecture. Moreover, an application is not necessarily designed to meet an application context as broad as supported by a data schema. Implementers are required to specialize and contextualize a generic data schema in accordance with implementation technologies and specific application requirements. These context-dependent requirements by high portability is difficult to be met or easy to be ignored by information models that typically have high extensibility. In general, requirements derived based on extensibility and portability are often unrelated, distinct and even contradictory, as described in Table 2.1.
<table>
<thead>
<tr>
<th>Extensibility</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>A broad application context</td>
</tr>
<tr>
<td>A specific application or a specific implementation</td>
<td></td>
</tr>
<tr>
<td>Concern</td>
<td>What the model is used for</td>
</tr>
<tr>
<td>How the model is (re)used</td>
<td></td>
</tr>
<tr>
<td>Structure</td>
<td>Composable structure</td>
</tr>
<tr>
<td>Comprehensible structure</td>
<td></td>
</tr>
<tr>
<td>Behavior</td>
<td>Design for extending</td>
</tr>
<tr>
<td>Design for porting (reuse)</td>
<td></td>
</tr>
<tr>
<td>Stage</td>
<td>Development</td>
</tr>
<tr>
<td>Operation</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 Different requirements on information models by extensibility and portability.

### 2.3 A model driven approach

The model driven approach focuses on integrating product data from different sources and reusing it for different application contexts, during the course of product realization. A system based on this approach should keep information in context, versionable, associable and retrievable in information models (Nyqvist, 2008). A model driven system interprets and processes data directly from information models rather than unstructured documents that are difficult to integrate and reuse. This mechanism relies on information models as a basis not only to represent and exchange information, but also to facilitate coherent interpretation and processing. The model driven approach is a typical composable solution that does not rely on context to an extent.

#### 2.3.1 Contextual proprietary information models

It can be observed that product data, as valuable intellectual resources in the engineering activities, is constrained by system that create it, as described in a tool-driven manner (Section 2.1.1). System vendors, instead of system users as data owners, control the ways to interpret, process and communicate the data. Accordingly, product data is modeled based on how the system is implemented, used and maintained. Schemas are directly based on internal data structure of the applications and thereby highly contextual. The produced data sets are typically proprietary lock-in, i.e. accessible only with specific applications or APIs provided by vendors. Although there might be APIs, a large number of details are hiding from users, which
makes it difficult to be reused by other systems. When the APIs are involved, it is useful to mention two types of programmability abstractions (Ford, 2013): Contextual and composable, with pros and cons to each. The common interfaces to current CAx systems are featured by the contextual abstraction, with the following advantages:

- High findability and feasibility for a specific problem.
- Concrete overarching templates.
- Being easy to learn and use.
- Being able to achieve specified user goals efficiently.

The benefits of the contextual abstraction apply well to information models. A contextual information model builds contexts into components, where it can provide “out of the box” data structure and methods for specific contexts that users search for initially. Thus, the contextual ones are easy to learn and use initially.

---

**Contextual abstraction and composable abstraction**

Ford (2013) proposed the two types of abstraction without formal definition. Abstraction can be seen as a selective technique to decide what should be visible and what should be invisible. There are different selective criteria to make the decisions. The criterion to distinguish contextual abstraction and composable abstraction is the implementation aspects. For an information model as an abstraction of product data, engineering activities using the data, i.e. application tasks, are an important concern. Another concern is about implementation activities. Different levels of abstraction according to existence of a concern about implementation activities can make a huge difference. Contextual abstraction takes the implementation context as a whole into account, which makes a large number of details in an information model related with implementation aspects. Composable abstraction does not consider the implementation aspects, which makes an information model contain few details about implementation aspects (Figure 2.24). In other words, the composable abstraction is independent of implementation details, so that it can be used to compose any implementation context with specific implementation aspects, which delivers high composability, but may require effort to make the composing happen.
However, when the information models are actually attached deeply with the systems, it is difficult to extend to other non-envisioned contexts, e.g. in other systems for similar problems. Besides, even though the solution is easy to learn and use, the users cannot reuse their experiences in other systems. Another major critique on the contextual approaches is the Dietzler’s Law (80% of what users want is fast and easy to achieve, 10% is difficult and expensive and the last 10% is impossible, but the users demand 100%).) when observing a project built upon Microsoft Access (Ford, 2008). The engineers required perfect artifacts to represent and convey their ideas, but the systems could only provide a nearly comprehensive solution through predefined processes and templates.

2.3.2 Composable standardized generic information models
On the other hand, standardized generic information models, as a basic enabler for the model driven approach, are a composable solution. They are decoupled from the contexts of specific systems. A standardized generic information model is not supposed to anticipate implementation details. The independency facilitates a high readiness for various contexts. It makes provision for possible use by different applications. Thus, a standardized generic information model can theoretically form a base to capture, integrate and represent information from different domains in different
contexts. For instance, the ISO standard STEP AP242 (ISO, 2014b) provides system neutral data structure that can be processed independent of applications and be reused by many different systems within a wide scope. When completely based on AP242, applications only provide functional support to accomplish engineering goals, rather than become technical limits to hinder communication.

Note that this definition of categorization for abstractions is relative. Compared with proprietary information models, ISO 10303 is more composable and less contextual, but specific definitions of syntax and semantics make it still contextual to some extent. The use of the “application context” suggests the semantic context. The syntax is usually formatted as an English-speaking-based “p21” file encoded in the ASCII standard. Hence, there are still some contextual prerequisites for a composable solution like STEP.

2.3.3 Combination of contextual and composable solutions

That said, the advantages of the contextual abstraction are compelling, which brings forward an idea of a new modeling architecture (Hypothesis I and section 3.5). The new modeling architecture adds a contextual layer to enhance the standardized generic information models without under-mining the original composable ability. The contextual layer represents information requirements for specific implementation methods. It makes information models fast to learn and use by a certain user group for a certain type of implementation tasks. Still the composable part should be kept to address unanticipated use cases. Users (i.e. the implementers) should be allowed to use any composable components together with the contextual layer.

The combination of the composable and the contextual solutions explains the strategy of the model driven system integration developed in this study. The existing composable standardized generic information models drive how applications can function effectively, the contextual layer of the modeling architecture drives how implementations can be performed efficiently. Different stakeholders can benefit from the model driven system integration in several aspects:

- Interoperability: Standardized generic information models are used to represent domain information used in applications and is supposed
to be decoupled from implementation practices. Hence, contexts of application are far more emphasized than contexts of implementation. As a result, standardized generic information models can be contextualized for different engineering applications without losing consistency of representation. That is, implementation strategies do not hinder interoperability of data sets.

- **Independency:** Models act as a medium for application users to interpret and process information as their intellectual assets. Any specific vendor or platform is not supposed to limit use of the models. Engineers should be allowed to choose when, where and how to manipulate information. Standardized generic information models facilitate a basis for envisioned applications to be integrated within an existing ecosystem. However, independency is also a major obstacle of development of standard communities, because it does not benefit major system vendors who have significant influences on the industry.

- **Productivity:** A major issue regarding product information in industry today is that information is processed, stored and communicated in a fragmented manner, which results in the tool-driven situation. Engineers’ workloads increase exponentially along with growing numbers of data sources, let alone inevitable information loss or errors during these efforts due to human errors and systematic incompatibility. Brunnermeier and Martin (2002) estimated imperfect interoperability costing about $1 billion annually and at least two-month delay for new model introduction, in the US automotive supply chain. Gallaher et al. (2004) quantified annual inadequate interoperability costs in US capital facilities industry as $15.8 billion. Therefore, as a basis of system integration, standardized generic information models have great potential to simplify the engineering activities and to increase productivity.

### 2.4 System architectures

It is a complex context that requires systems to be architected before detailed design and construction. Implementing system integration is naturally an interdisciplinary task, especially to satisfy complex but critical
business needs and technical needs in the engineering context. An architecture should directly address concerns of those stakeholders with distinct expertise who are inevitably involved in an entire lifecycle of an integration application. The stakeholders are a central part of the complex context, who possess knowledge, perform tasks and interact with systems. There are immediate stakeholders, e.g. designers, programmers, testers and users, who are most concerned about technical tasks, in terms of design, development, evolution and maintenance. There are also non-technical stakeholders such as customers and organizational managers who are indirectly influenced by system performance to accomplish high-level business needs. All the stakeholders are living in a constantly changing environment where technologies and business change frequently. The coordination of technologies, business needs and concerned stakeholders imposes great requirements on architectural design to deal with the changing circumstances.

An architecture defines overall system structure with a set of design decisions (CMU, 2016) which specify selection, organization and composition of architectural elements (Booch, et al., 2005), in terms of forms and rationales. An architecture mainly addresses concerns about functional requirements and quality requirements of systems. Quality requirements drive development of system architecture, especially for software-intensive systems (O’Brien, et al., 2007). The previous sections of this chapter have revealed a quality requirement mostly relevant for system integration, i.e. interoperability. An architecture targeted on the interoperability is critical for overall performance of system integration.

<table>
<thead>
<tr>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>An architecture, as a term used in many domains, describes underlying structure of a system (Brand, S., 1995). Architects propose solutions with balanced integration of durability, utility and beauty (Vitruvius, 1914). The ultimate goal of architecture is to manage complexity at an understandable level for concerned stakeholders. It is usually not necessary to draw 3D design, to plan a budget and to set up a schedule to build a simple doghouse (See the figure below). However, it is impossible to accomplish a complicated building without architecting to orchestrate an enormous number of factors and correspondences. An architecture helps people to reuse previous knowledge of methodology to solve problems at</td>
</tr>
</tbody>
</table>
hand, to reduce complexity with reusable design principles and to fulfill certain requirements (Spinellis and Gousios, 2009).

An architecture of an information system also shares a similar definition. ISO/IEC/IEEE (2011b) defined an architecture as “fundamental concepts or properties of a system in its environment embodied in its elements, relationship, and in the principles of its design and evolution.” It can represent a process from requirements to a final developed system (Bass, et al., 2012). Description of an architecture shall be a core work product to enable communication and cooperation among stakeholders (ISO/IEC/IEEE, 2011b).

Several basic techniques are important for successful architectures. Modularization with prudent abstraction forms a major means to reduce system complexity and increase architectural agility. Stakeholders need modules to understand, develop, use, or maintain the systems. All the major architectural patterns, e.g. layered, event-driven, service-oriented and pipelines, are about modularization. The most fundamental principle for architecting is high cohesion within a module and low coupling between modules, which leads to concerns of other internal quality attributes for implementation, i.e. reusability, readability, maintainability and changeability. Examples of modularization of an architecture are a layered pattern
describing a modeling architecture in Section 3.5 and an MVC (Model-Viewer-Controller) pattern to specify an integration architecture in Section 4.9.

<table>
<thead>
<tr>
<th>Architecture pattern and architecture style</th>
</tr>
</thead>
</table>
| Changes happen all the time during an entire system lifecycle. Decisions on a system architecture are all about compromise. That is, a change in one feature often impedes another feature (Spinellis and Gousios 2009). It is a good design of a software architecture that guarantees changes to cause minimal impediments. Patterns provide general answers on how to reach good designs. Alexander (1979) defined a pattern as a three-part statement: “A certain context, a problem, and a solution.” It is the patterns that present well-proven schemes guiding architecture design for recurring problems in certain contexts (Buschmann et al., 1996).

A style is a descriptively and prescriptively codification of a set of architectural design solutions with certain commonality (Perry and Wolf, 1992). Unlike patterns, such codification is often incomplete for any specific problems (Bachmann, et al., 2011). Similar to styles of building architecture, software architecture styles are more useful for categorization and documentation, rather than problem-solving.

In particular, the MVC (Model-View-Controller) pattern is chosen for its specialized suitability for interactive systems with heavy needs of coordinating data representation and presentation. The MVC pattern is a decent way to achieve loose coupling between data models and UIs (User Interfaces). It highlights reusing data for different contexts as a key feature of any interactive system. An information model is usually easy to be presented where it is created, but can be useful only when reused, integrated and presented in other contexts. MVC is a basic infrastructure to facilitate reusing of data. This thesis will present how to apply the MVC pattern to implement the information models as a basis to integrate and reuse engineering data in Section 4.9.

An implementation process delivering stage-by-stage architectural decisions is also challenged in this particular context. Formal information models are often specified with syntax uninterpretable by human beings, which affects implementation for readable presentation. Fortunately, the
interpretation of the information models is possible for computers. Thus, there should be a specific implementation guidance to translate the computer-interpretable representation to human-interpretable presentation. Common development solutions like agile development methodology with iteration based on prototypes or increments is often criticized for insufficient predictability and high expense, and, on the other hand, Normal waterfall-style sequential development process is too rigid to adapt changing requirements and inadequate design in reality. It is promising to compensate these methodologies with contextual interpretation of models, as proposed by the Hypothesis II, to reach a system architecture design with relatively low complexity and high implementability.

2.5 Information architectures

In an information model, a large amount of information is usually integrated according to a specific scope and related requirements regardless of usefulness for particular processes or applications. It has to be done this way to keep completeness and consistency of the data and thereby to ensure high readiness for different applications. The mechanism is guaranteed by regulated syntax, description methods, interpretation methods, implementation methods and validation methods. The resultants of completeness and consistency contribute to high semantic interoperability and high computer interpretability. Naturally, human interpretability of information models is ignored by the complicated mechanism. Let along data amount is often too large to be cognitively processed.

The major objectives of digital technology are to augment human intellectual abilities and to compensate human limitations in practice (Bush, 1945), rather than to stand in the way. In the engineering context, the augmentation is greatly influenced by data generated through a full product life cycle, which can be used for many purposes: To automate processes, to simplify activities, to convey knowledge, to evaluate ideas and to make decisions. Information models make the data interpretable by computers, but it is the responsibility of digital applications to make the data applied effectively for the contextual purposes, i.e. to augment human intellectual abilities. In fact, similar stories have happened everywhere in contexts of integrating and sharing information. Data should not just be organized to make applications work well, but also to facilitate ease of use by human
users. Thus, a good interaction design to present information, guided by an information architecture, should be delivered to let information make sense to users.

ISO/IEC (2010) defined *information architecture* (IA) as “(human-centred) structure of an information space and the semantics for accessing required task objects, system objects and other information.” Practicing IA is all about organization and description of content in terms of objects or items in a principled space. Patterns and sequences are two most important aspects to organize and describe content (Garrett, 2010). There are two general objectives of designing IA: 1) Help users accomplish their goals by easily navigating content and locating objects and 2) Help vendors accomplish business goals by educating, persuading and notifying users.

Today, UX (User Experience) designers for everyday commercial applications usually consider IA as a part of user experiences, which is regular but not dominant. After all, it is not worthy to put much effort on architecting information that is not so sophisticated, when attracting and engaging users is more prioritized. However, most engineering applications are characterized by overloaded information, burdensome cognitive tasks and demanding domain expertise (Redish, 2007). These complex professional systems usually have a small user group and are rarely encountered by the HCI (Human-Computer Interaction) experts. Research on complex systems obviously lacks economy of scale and, therefore, the HCI community is less likely to consider such scenarios.

Engineering CAX systems are typical content-heavy applications. Information from different sources in different forms are required to be integrated and to be used for many kinds of processes. As implied by the naming, at most time CAX systems aid engineers to accomplish some goals, rather than to perform the jobs automatically. The aid starts with visualization of overloaded information and ends with a conveyer to communicate information with other systems or other human users. The troublesome tasks with high requirements on efficiency and effectiveness make engineers lack time to learn patterns of interaction and presentation as they might succeed in normal applications. Hence, architecting information can provide a way for users to easily interact with information with organized and clarified content and processes (Spencer, 2010).
The overall system integration is facilitated at different levels. Applying information models is not only connected with specific systems, but also connected with specific business logics and workflows in an interactive environment. At a business level, a system integration solution is heavily determined by qualities of interaction, e.g. findability, usability and learnability. Standardized representation enables interoperability at the levels of syntax and semantics. In system integration, properly architecting information improves interoperability to a structural level. At this level, the information representations are presented with patterns and sequences to ease the human interpretation.
Chapter 3
A new modeling architecture

All problems in computer science can be solved by another level of indirection.
- David J. Wheeler

This study aims at a practical solution for efficient system integration to overcome the tool-driven manner (Section 2.1.1) and to effectively support product realization. The solution should support conventional implementation practices and fix the efficiency issue caused by the focus on application contexts (Section 3.2) when developing information models. It requires a new modeling architecture (Section 3.5) to facilitate high pragmatic interoperability (Section 3.1). Implementation contexts (Section 3.3) and implementation models (Section 3.4) are new concepts introduced in this study as imperative parts of the new modeling architecture.

3.1 Envisioned pragmatic interoperability

Anything for communication has its semiotic aspects. In the Information Age, syntax may remain as a stable concept, but semantics and pragmatics have been evolving with the employment of computers (see Section 2.1.3). Within this semiotic framework, computer interpretation requires a decent semantic level. As introduced earlier (Section 2.2) semantics of information models has been defined very well. Ideally, model developers create semantically perfect data schemas so that all conformant data sets can be automatically processed and effectively reasoned upon by computers.

Nevertheless, human interpretation of the information models is still inevitable, especially for implementation, which requires pragmatic interoperability of the information models as well. Griffin, et al., (2002) proposed the highest level of data interoperability as seamless sharing of information, in a form of common exchange structure as well as universal
interpretation. The human use of information models should not only focus on representation for effectively capturing and integrating information from different sources, but also on ease of human use. However, existing interfacing techniques do not support human interpretation sufficiently. Thereby, enhancing the pragmatic interoperability is the focus on the new modeling architecture.

This study claims that the implementers are the core stakeholders for human interpretation of information models. It is critical to emphasize the human audience when studying the pragmatics of any kind of representation. Superficially, pragmatic interoperability of an information model should be facilitated for scenarios of both development (with data schemas) and use (with data sets) of applications. In other words, implementers can easily use data schemas and application users can easily use data sets. Figure 3.1 illustrates the use of the two types of information models (data schemas and data sets). Figure 3.1 also shows that an application is an unavoidable media for use of a data set and the application users actually do not interpret any form of information models directly. Applications, developed by implementers, control all pragmatics for application users. Therefore, in fact, the implementers perform the only direct human-model interaction.

![Figure 3.1 Pragmatic interoperability that should be addressed by information models.](image)

The implementers’ role hereby makes implementability of an information model, for either system integration or new functional development, exhibit pragmatic interoperability. Taking kinematics as an example, kinematic modeling (paper E) and kinematic error modeling (paper A) have been studied and implemented based on STEP. Within the scope of
STEP, the first edition of a data schema for kinematics was published as a resource with decent completeness and validity in 1996 (ISO, 1996); AP214 (STEP Part 214: Application protocol: Core data for automotive mechanical design processes, ISO, 2010a) was the only application protocol integrating it; the first published data set was done by Hedlind, et al. (2010) and the first published conformant application was done by Li, et al. (2011); no commercialized application has been developed until the second edition of the data schema was published (ISO, 2014a). It might be difficult for average implementers to use the data schemas as what Hedlind, et al. (2010) or Li, et al. (2011) did.

Same things happen when implementing new functions (rather than just integration): No one would expect a CAx solution for kinematic error analysis, without high implementation readiness of related information models (paper A); no one would expect digitalization of product catalogue communication, without high implementation readiness of standardized digital catalogues (paper C).

Not only implementability related, portability and usability of information models are also in relation to pragmatic interoperability. As introduced in Section 2.2.3, portability is an important quality yet not very well satisfied and high pragmatic interoperability of an information model can lead to high portability. With a modeling architecture to facilitate human interpretable information models, it is easier for implementers to use data schemas in new contexts, which means easier to “port” the data schemas in new contexts. In turn, high portability exhibits high pragmatic interoperability. Moreover, in its own right, portability of an information model is equivalent to ease of implementation (implementability). From the perspective of implementers as users of an information models, portability is also equivalent to usability in this context. Thereby, with implementers as direct users, pragmatic interoperability of an information model facilitates its portability, usability and implementability (Figure 3.2).
3.2 Application contexts: A pitfall

Any architecture shall include a correct definition of a context as an initial part, so shall a modeling architecture. From a semiotic perspective, a context is where interpretation of an artifact happens. Understanding of the context helps developers know what kinds of artifacts are best for interpretation in the specific environments.

3.2.1 Application contexts ignoring implementers

Section 2.2.1 has introduced application contexts as a current initial point for both software development and information model development. The application contexts help developers in both domains know the interpretation needs of application users. Particularity for information model development, understanding of the application contexts has been beneficial for standardization of semantics (more details in Section 2.2.2).

Nevertheless, is an application context sufficient for initialization of information model development (Figure 3.3)? The answer is no, in that the implementers, instead of application users, become the direct and primary user group of the final products (data schemas). The final products are not used in the application contexts. Hence, the application contexts fail to represent the environment where the final products are used. In a word, using application contexts as the initial part of a modeling architecture violates the reason to specify contexts for development of an interactive system.
3.2.2 A different implementation workflow

This misplaced focus on the application contexts leads to difficulties faced by implementers. One consequence is that involvement of information models makes an implementation workflow significantly unconventional.

How to implement an information model has been introduced in Section 2.2.2. Taking ISO 10303 STEP as an example (Figure 3.4), model developers should generalize all possible use cases in application contexts and map into a comprehensive list of information requirements. This mapping in model development is performed between the AAM and formalized ARM, in order to create a possibly very huge AIM for a generic information standard.
Figure 3.4 The traditional way to implement an application conforming to a STEP data schema.

Hence, for implementers, locating a valid segment of a data schema (e.g. the AIM) is an unavoidable task. The task needs AAM and ARM as intermediates to manually interpret AIM (arrows from the implementer in Figure 3.4). Moreover, the model artifacts (AAM and ARM) convey information (e.g. contexts and requirements) that may be familiar for implementers, but the syntax is unfamiliar. The difficulty is undoubtable to consistently map one unfamiliar representation to another (Gielingh, 2008).

This mapping mechanism makes the implementation process greatly entangled with the information model development process (illustrated by the two biggest blocks in Figure 3.4). These practices are neither intuitive nor efficient for implementers who are direct users of model artifacts in a modeling architecture. By manually interpreting most model artifacts, implementers, as users of a system (i.e. the AIM), practically learn how this system is developed. This manner is not reasonable for design of any kind of interactive systems. Just like few software users can understand how software is designed and developed, few implementers can do the same for information models.
3.2.3 Increasing complexity of modeling architectures

Besides, each generation of the STEP-based modeling architectures tends to increase the number of model artifacts (Figure 3.5). The evolution in Figure 3.5 suggests that more and more layers and modules are involved for each modeling architecture. It is caused by a critical need to support rapid model development in the changeable application contexts (Section 2.2.4). Hence, model developers need a flexible structure to facilitate rapid standardization and backward compatibility, i.e. 1) extensible definitions of the application contexts, 2) reusable semantic constructs and 3) standardized syntax with designated access interface. Eventually, the use of data schemas (e.g. the AIMS in Figure 3.5) are instructed with complex hierarchical structures.

Figure 3.5 Evolution of the modeling architectures, based on ISO (2016c).

It results in more and more complex structures applied in implementation projects. These structures are problematic for the implementers as direct users and interpreters. Implementers need to learn different mapping mechanisms when different data schemas are applied. They need to interpret piles of documents to specify information models at different levels of abstraction. The appended paper D describes a case study for this issue during implementation.
3.3 Implementation contexts: The new concern

Therefore, an intended context where an information model is used should be sufficiently defined in the modeling architecture which specifies development of the information model. In general, a context should describe “relationships, dependencies, and interactions between a system and its environment” (Rozanski and Woods, 2012). For an interactive system, the context should mainly describe the interaction between primary users and the interactive system. This is why an application context can form a basis to develop applications. However, for an information model, the primary user is who use the data schema directly, i.e. the implementers, in that any use of information models begins with implementation.

3.3.1 Defining implementation contexts

With this knowledge, the context is where implementers use an information model to implement an application, which is named as an implementation context. Note that the information model in this definition means standardized generic information models interacting with implementers with model artifacts, e.g. data schemas, reference models and data interfaces; examples are widely accepted Application Protocols (APs) of ISO 10303. Like application contexts, an implementation context is a part of a modeling architecture where it guides development of information models by describing the interaction between implementers and information models (Figure 3.6).

![Figure 3.6 The usage of the implementation context.](image)
For a modeling architecture, an implementation context includes implementers and applications (the implementers’ primary goal) as major components (Figure 3.7). Identification of implementation contexts addresses the implementers as the primary users of information models, in development of modeling architectures. This identification of an appropriate audience is a basis of any discussion about pragmatics and the context of the pragmatics. Any developer shall clarify this identification to know whether their creation is useful and usable. As a result, a modeling architecture guided by an implementation context can deliver data schemas appropriate for implementers.

![Figure 3.7 Implementation context, a shifted basis where information model development begins.](image)

As also illustrated in Figure 3.7, the originally-defined application context is a part of the implementation context. This is a result of taking into account the entire software implementation process where implementers aim at problem solving in the application context. Including application contexts to a certain detailing level is how implementation contexts are
concerned with indirect elements. Comparably, a typical application context also includes indirect elements (e.g. needs, characteristics, tasks and knowledge of the customers).

The inclusion of application contexts suggests that boosting portability of an information model shall not jeopardize its extensibility (more details in Section 2.2.3). It also suggests a layered structure where one layer encapsulates another for different purposes. The original modeling architectures are kept to maintain extensibility, but a new layer should be added to facilitate portability. This new layer is what will be introduced in Section 3.4.

To describe the implementation context in detail, the following subsections will make a categorized discussion about the relevant contextual components based on paper D. The categorization is based on instructions provided in ISO 25063:2014 Common Industry Format (CIF) for usability: Context of use description. However, these subsections begin with a term “system-of-interest” that actually cannot be categorized as a contextual component.

3.3.1.1 The system-of-interest
This study adopts the concept system-of-interest agreed upon by several ISO standards, as “the system whose life cycle is under consideration in the context.” It is not a part of the context, but it is the centric subject for which the context is investigated. All the relevant components in the context influence the system-of-interest. For an implementation context, a system-of-interest is a modeling architecture with a goal of underpinning implementation of model driven system integration. The modeling architectures shall support implementation (in an implementation context) in addition to processes and information flows (in an application context). Unlike the traditional modeling architectures based on application contexts focusing on supporting information model development, this also supports information model use.

Hence, the subject domain of an architecture is the software-intensive system implementation, more specifically, for a system aiding engineering activities in the process of product realization. The generic information
models, the formal data exchange structure and the standardized data access interfaces defined within the modeling architecture of ISO 10303 STEP facilitate a primary part of the architecture.

3.3.1.2 Users
Technology enhances human capabilities to address human needs. When it comes to IT, the human capabilities need to be enhanced to take control over the technology in turn (Syväjärvi, et al., 2005). To take advantage of information models, three major types of human users of IT systems are involved: Application users, implementers and model developers. These three user groups are introduced and discussed in detail in the second section of the appended paper D.

This categorization is also valid in implementation contexts (Figure 3.8). The implementers are direct and primary users in the implementation contexts, initiate use of information models in implementation contexts. The implementers are concerned with the application users, as well as customers in Figure 3.8, during their implementation activities. The application users and the customers can be two adjacent players in a supply chain who need to exchange data represented in a data set, or can be two adjacent roles in a workflow in the same organization, e.g. a product designer and a process planner. The model developers should be aware of the implementation contexts, and are secondary users in the implementation context. It means that they may be applied for maintenance tasks.
This categorization is based on roles, but should not be applied to a specific person because a person can easily switch her/his role depending on the task the person is taking. For instance, most model developers are also industrial practitioners, who can be application users or customers. End-user developers (Lieberman, et al., 2006) are also a widely observed role in the world of manufacturing engineering, who can switch between the application users and the implementers in different scenarios.

3.3.1.3 Tasks
Tasks are represented by a series of interaction activities that can form a hierarchical structure (Annett and Duncan, 1967). Defining a task is essentially context dependent. Users’ goals and available technologies often determine content of tasks. Taking CADCAM integration for example, there seems to be a kind of linear-distributed, phase-oriented allocation of tasks as displayed in Figure 3.9, but the truth is that a structured engineering process cannot guarantee a successful solution, while a flexible style is recommended by most experienced mechanical designers. In the appended paper B and E, the tasks in implementation contexts have been discussed in two system integration projects for cutting tool classification and kinematic data exchange respectively.
Particularly for a context of engineering use of technologies, at a micro level, a pattern can be generalized with a cycle of analysis, synthesis and evaluation. Engineers in either software industry or manufacturing industry are used to shifting across these three modes and applications should provide support for these modes and the shifts. Analysis as a cognitive process requires presentation and visualization of knowledge in a structured way. Synthesis as mainly a motor process requires aids on intuitive integration and manipulation of information. Timely feedback is essential for evaluation. The shifts between these actions need seamless integration of functions and data, because most design breakthroughs happen during rapid shifting which is mostly influenced by cognitive activities of engineers. In general, information technology should provide sufficient support for the cycle and it is the highly extensible and portable standardized information models that lead to an opportunity to help engineers in different domains to complete this cycle effectively and efficiently.
3.3.1.4 **Technical environments**

The technologies that support system integration in an engineering context can be categorized as two types: Implementation platforms and information models. An implementation platform is usually a function-oriented package binding with a designated IT environment and with dedicated interfaces for implementers. In the third section of the appended paper D, a thorough discussion is made for the state-of-art implementation platform based on standardized information models.

It is the information model that establishes infrastructures for the implementation platforms and the implementation contexts. Semantic interoperability (independent of applications and implementation processes) is often a major goal to deliver standardized information models. The portability issue of the information models originates from this goal and is difficult to be tackled by normal implementation platforms that simply comply with the standardized data access interfaces, e.g. SDAI (Standard Data Access Interface, ISO, 1998a).

In an application context, information technologies are supposed to augment the human intellectual ability and produce artefacts accurately, quickly and easily. However, currently most engineers are “kidnapped” by technologies, i.e. in a tool-driven manner. The data they create, manipulate and communicate is limited by applications. Consequently, organizations in manufacturing industry drives employees, suppliers, IT supporters partially based on the software applications they are using. Hence, an important contribution of the model driven approach is to return the ownership of information back to engineers. By widely-accepted, vendor-independent, standardized modeling technologies, users are allowed to control their information assets in any way they prefer and to convey their knowledge (requirements, specifications, designs, reasoning, etc.) accurately with others.

3.3.2 **Use of implementation contexts**

Figure 3.10 illustrates position of the implementation context in the modeling process. This illustration is an update of Figure 3.4 due to the involvement of the implementation context, and still takes STEP as example. In
the illustration, the implementation context is used to define a set of information requirements for the use of a data schema. The meanings of this illustration can be obtained by row and column. Rows group specific processes and columns group similar levels of abstraction.

Figure 3.10 In the information model development process.

The original standardized model development process is simplified into the second row in Figure 3.10. Model artifacts less useful, e.g. AAM (Application Activity model) and ARM (Application Reference Model), are omitted only for simplification of illustration. To avoid confusion, the original information requirements are renamed as standardized information requirements in Figure 3.10. One result of this process is still the AIM (Application Interpreted Model) which is not recommended to be used by implementers directly. Instead, an implementation model encapsulates the AIM and is realized by an API for implementers. The API is also an encapsulation of the traditional interface techniques, i.e. SDAI and conformance testing tools. The process to create the implementation model and the API is the first row of Figure 3.10. Section 3.4 will introduce how to specify a new set of information requirements and the implementation models. Section 5.1 will use a case study to exemplify the creation of the API. The third row and the fourth row depict a software implementation process with artifacts that are relevant for this study.
Columns of Figure 3.10 represent groups with similar levels of abstraction. Contexts and requirements formulate the first and the second columns to demonstrate their imperative roles in any development process. Between the contexts and the requirements, the only type of relationship is aggregation. It suggests inclusive relationships, e.g. between the implementation context and the application context. Consequently, the following artifacts also have same relationship. Then, the third column represents artifacts satisfying the requirements.

Similar to any context, the usage of implementation context is to generate requirements. For information models, the set of requirements are still called information requirements. To develop a new modeling architecture, information requirements still describe 1) what should be standardized in a data schema; 2) what should be included to use the data schema during a use process (i.e. implementation). From a semiotic perspective, the former is at a semantic level, which can be facilitated in the original modeling architectures; the latter is at a pragmatic level and will be introduced in Section 3.4.3 for creation of an implementation model.

**Software implementation process, implementer and implementation context**
Implementers perform implementation based on an application context through a *software implementation process* “to produce a specific system element (software item) implemented in software.” (ISO/IEC/IEEE, 2015) During a software lifecycle, an information model can be shared, ported, or reused. The software implementation process, initiated by diverse technical needs and business needs, that results in heterogeneous forms of application software and in great complexity of software engineering practices.

The process is performed by *implementers* who are individuals or organizations that perform implementation tasks (ISO, 2008). The implementers may have different understanding in data, functions, contexts, etc., with business and technical constraints. This fact leads to differences in design, construction and deployment processes and the differences in delivered software functions and capabilities. All these differences make it not easy to create an implementation-oriented artifact.
This study introduces the term `implementation context` to define the environment where implementers develop software application using information models. In this thesis, the implementation context is delimited as the intended context of using an information model. However, the concept of “implementation context” can be generalized for any subject directly used by implementers, which is out of the scope of this thesis.

Both implementation contexts and application contexts will be translated to information requirements and information models. Differences between the products from the two contexts lie in detailing levels. The implementation context needs an information model specified in accordance with how it will be used, i.e. to support implementation, application, and maintenance. On the other hand, the application context needs an information model specified regardless of implementation details, but only for the use of applications, i.e. information flow and processing during engineering activities. Implementation contexts require less detailed semantics and more detailed pragmatics. Therefore, the produced information requirements and information models should represent these requirements.

### 3.3.3 Discussions of implementation contexts

There are several benefits by introducing implementation contexts for both information models developers and implementers. 1) The implementation contexts are an effective base for developing an information model friendly for its users, which boosts the pragmatic interoperability. 2) The software implementation process will not be entangled with the model development process. All relevant artifacts have been encapsulated in APIs as the only interface for implementers. Just like the construction stage in a normal implementation process, implementers choose APIs for different components according to software design. 3) Information model developers and implementers can reach a consensus on modularization of APIs by understanding implementation contexts. As introduced in Section 2.2.2, Implementers should select useful parts of data schemas for implementation. The modularization is a technique to facilitate the selection in a way more familiar to implementers. Exemplification of the modularization of APIs can be found in Section 5.1 and paper D.

ISO/IEC/IEEE (2011a) regarded implementation free as an important characteristic of individual requirements and this principle is not violated.
here. Cooper, et al. (2014) considered the main principle for requirement
definition is to “define what the product will do before you design how the
product will do it.” Does the focus on implementation context diminish the
quality of the information requirements? Note that the system-of-interest
when defining information requirements is the information model, more
specifically, the data schema. The information requirement is introduced
to develop information models, rather than to be implemented in an appli-
cation. Hence, a set of information requirements for an information mod-
els may address detailed information about application implementation
(not model development) without hurting its implementation independ-
ency. The information requirements should be free from methods to de-
velop an information model.

3.4 Implementation models

The implementation model is proposed as an information model that en-
capsulates one or more standardized data schemas, with additional sup-
portive functions for data set manipulation, to satisfy information require-
ments in an implementation context.

3.4.1 Semiotics of information models

Creation of an implementation model requires an in-depth understanding
of an implementation context (Figure 3.14), of which information models
are actually the most confusing concept for all stakeholders. It is therefore
critical to fully clarify this concept as a part of implementation and appli-
cation. An information model is used for communication with a process of
representation and interpretation, which is a “sign process”, i.e. “semiosis”
defined by Peirce (1907). Human interpretation, i.e. pragmatic interoper-
ability, has been identified as a gap to use information models. Hence, it is
useful to define and categorize information models from the perspective of
semiotics, which will lead to the exact problematic parts.

Data sets and data schemas are two information models mostly re-
ferred in this study and are in direct relation with stakeholders. They are
mainly used by application users and implementers respectively. A data set
conforming to a protocol can be roughly treated as an instance of a (part
of) data schema of the protocol. For a normal application protocol in STEP,
three semiotic aspects of data sets and data schemas are facilitated by different specifications (Table 3.1). This semiotic perspective can lead to in-depth understanding of current problems and practical needs.

<table>
<thead>
<tr>
<th></th>
<th>Syntax</th>
<th>Semantics</th>
<th>Pragmatics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set</td>
<td>ISO 10303-21,</td>
<td>an AIM of an AP</td>
<td>UI specified by implementers</td>
</tr>
<tr>
<td></td>
<td>ISO 10303-28, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data schema</td>
<td>ISO 10303-11 (EXPRESS)</td>
<td>AAM/ARM, etc.</td>
<td>AP documents, ISO 10303-22 (SDAI), ISO 10303-27, etc.</td>
</tr>
</tbody>
</table>

Table 3.1 Specifications to facilitate semiotic aspects of information models standardized in the framework of STEP.

Data sets are application-user-oriented. For example, a structurally complete STEP data set conforming to ISO 10303-21 is displayed in Table 3.2. The syntax of all the lines of Table 3.2 is governed by ISO 10303-21; this syntax of data set can be reformatted to XML governed ISO 10303-28 (ISO, 2007b). Semantics of a data set is specified by a data schema, i.e. each instance in a data set, exemplified in Table 3.2, instantiates an entity in the AIM of ISO 10303-242; the complete data set is valid according to the AIM. Pragmatics of a data set is ideally determined by UI of applications (and practically by implementers). UIs are designed to display different facets of a data set, according to designated tasks. For example, Table 3.2 shows an excerpt of the data set for the kinematic link. Users of Siemens NX can explicitly discern a kinematic link with its corresponding geometry (Figure 3.11) represented in the data set; users of STEP-NC Machine interpret the kinematic links with motion simulation (Figure 3.12); users may also interpret a data set in a STEP model translator without 3D visualization (Figure 3.13).
Table 3.2 An excerpt of a p21 (conforming to ISO 10303-21) data set.

<table>
<thead>
<tr>
<th>#2695=KINEMATIC_LINK('base');</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2696=RIGID_LINK_REPRESENTATION('',(#2736,#2747),#2665,#26695);</td>
</tr>
<tr>
<td>#2697=KINEMATIC_LINK_REPRESENTATION_ASSOCIATION('','',#2696,#176);</td>
</tr>
<tr>
<td>#2698=PRODUCT_DEFINITION_RELATIONSHIP_KINEMATICS('',$,#33);</td>
</tr>
<tr>
<td>#2699=CONTEXT_DEPENDENT_KINEMATIC_LINK_REPRESENTATION(#2697,#2698);</td>
</tr>
</tbody>
</table>

Figure 3.11 UI of Siemens NX 7.5 visualizes a kinematic link in relation to its corresponding geometry.
As defined in Table 3.1, a STEP data schema (e.g. an AIM) also has its semiotic aspects facilitated with a different set of techniques. For example, the instances Table 3.2 are governed by the AP242 schema partly listed in Table 3.3, where the textual descriptions are based on ISO 10303-105:2014. For such kind of data schemas, ISO 10303 STEP defines EXPRESS as a modeling language to govern the syntax. The EXPRESS is also reused in other standards, e.g. ISO 13584 PLib, ISO 14649 STEP-NC and ISO 16739 IFC (Industry Foundation Classes). Semantics are governed by AAM and ARM that respectively specify application contexts and corresponding information requirements at different levels of abstraction. Based on AAM and ARM, model developers should designate meanings to the data schemas. This designation process for an AP is called “interpretation” (ISO, 1994). Pragmatically, a verbose “dictionary” (AP documents) and a “gram-
mar book” (SDAI and programming language bindings) are given to implementers to interpret the data schemas. These tools are insufficient because they are actually presented only at a semantic level and a syntactical level, respectively. These tools cannot ensure efficiency and effectiveness of human interpretation.

SCHEMA ap242ManagedModelBased3dEngineeringMIMlf;
...
(*
A context_dependent_kinematic_link_representation is the association of a kinematic_link_representation_association with a product_definition_relationship_kinematics. The kinematic_link_representation_association identifies the shape of a kinematic_link_representation as it plays the role of the related_product_definition in the product_definition_relationship.
EXPRESS specification:
*)
ENTITY context_dependent_kinematic_link_representation;
  representation_relation:
    kinematic_link_representation_association;
  represented_product_relation:
    product_definition_relationship_kinematics;
END_ENTITY;
(*
Attribute definitions:
representation_relation: a kinematic_link_representation_association that is associated with the product_definition_relationship_kinematics.
represented_product_relation: a product_definition_relationship_kinematics that identifies the shape of the related kinematic_link_representation_association in the context of a product_definition_relationship.
*)
...
(*
A kinematic_link is a type of vertex representation of the topological aspects associated with a rigid part of a mechanism.
EXPRESS specification:
ENTITY kinematic_link
  SUBTYPE OF (vertex);
END_ENTITY;

(*
A kinematic_link_representation_association is a type of representation_relationship that associates a representation with a kinematic_link_representation.
NOTE    A kinematic_link_representation_association is used to define the shape of a link by associating a shape_representation to the corresponding kinematic_link_representation.
EXPRESS specification:*
*)

ENTITY kinematic_link_representation_association
  SUBTYPE OF (representation_relationship);
  SELF\representation_relationship.rep_1 : kinematic_link_representation;
  SELF\representation_relationship.rep_2 : shape_representation;
WHERE
  wr1:
      context_of_items ) OR
    ('AP242_MANAGED_MODEL_BASED_3DENGINEERING_MIM_LF.' +
    'REPRESENTATION_RELATIONSHIP_WITH_TRANSFORMATION' IN
    TYPEOF( SELF ) ) );
END_ENTITY;

(*
Attribute definitions:
rep_1: the kinematic_link_representation with which rep_2 is associated.
rep_2: the shape_representation with which rep_1 is associated.
Formal propositions:
WR1: The context_of_items of rep_2 shall be identical to the link frame of the kinematic_link_representation with which rep_2 is associated.*)
A product_definition_relationship_kinematics is a type of property_definition. The product_definition_relationship_kinematics specifies the kinematic property of a product relationship.

EXPRESS specification:

*)
ENTITY product_definition_relationship_kinematics
  SUBTYPE OF (property_definition);
  SELF\property_definition.definition : product_definition_relationship;
  UNIQUE
    ur1: definition;
END_ENTITY;
(*)

Attribute definitions:
Definition: an inherited attribute that shall be of type product_definition.

Formal propositions:
UR1: The definition shall be unique within a population of product_definition_relationship_kinematics.
*)
...
(*

A rigid_link_representation is a type of kinematic_link_representation.

EXPRESS specification:

*)
ENTITY rigid_link_representation
  SUBTYPE OF (kinematic_link_representation);
END_ENTITY;
...
END_SCHEMA; --
p242_managed_model_based_3d_engineering_mim_lf

Table 3.3 An example of an ISO 10303 data schema (a part of the long-form schema of STEP AP242).
From the semiotic perspective, the aim of this study is to use implementation models in a modeling architecture to effectively facilitate pragmatics, i.e. human interpretation of data schemas. Data sets do not concern this study. The problems in terms of semiotics have been clarified previously: 1) Implementers (i.e. interpreters) have to generate semantics of data schemas, which should not be an interpreters’ task and 2) the designated access interface (SDAI) of data schemas is not sufficient to interpret the data schemas. Hence, implementation models should satisfy 1) suitable abstraction of semantics for implementation and 2) specification to facilitate interpretation at a pragmatic level. These two requirements are underlying principles to define a new modeling architecture.

3.4.2 Encapsulation

As implied earlier, encapsulation is crucial for the creation of an implementation model, i.e. defining what should be exposed and not to the implementers. The nature of high semantic completeness and high extensibility of a data schema makes not all its elements useful for a particular implementation. Thereby, data schemas are often tailored in practice because implementing an entire application protocol is usually not necessary. Only those relevant for implementation shall be exposed in the implementation models. Moreover, entities in data schemas only concerned about application domains are not sufficient for implementation, i.e. a lack of functionalities.

As defined previously, these facts are managed by an implementation model with encapsulation is performed in three steps: 1) Completeness and granularity of existing verbose data schemas are properly managed and abstracted; 2) additional functionalities to complete implementation tasks are included; 3) entities and functionalities are communicated in a pragmatically effective manner. The derived new set of information requirements should reflect the first two steps of this encapsulation anyhow. The third step is about pragmatic use of an implementation model, i.e. application programming interfaces (APIs) should be realized.

It is still important to keep the existing modeling architecture and the existing implementation process based on the application contexts. The encapsulation make some content in data schemas invisible but does not abandon or replace the invisible content. This means that this solution
should provide a wide range of abstraction levels to facilitate different users’ needs in implementation contexts. In particular, there should be a mechanism to bypass the encapsulation and go back to the original way. This mechanism is presented as a layered architecture for system integration in Section 3.5. There are several benefits from the mechanism: 1) Implementation models is not compulsory and can be developed for a part of a data schema. 2) Existing implementers are also free to reuse their knowledge to implement the traditional information models. 3) Implementation projects should be allowed to switch implementation styles between the new one and the old one to meet the needs of different implementers.

3.4.3 Information requirements

The creation of an implementation model will be presented in the next subsection, but as a critical step, the creation of information requirements is presented here in a separate subsection. An implementation context is an initial point for the creation of an implementation model; the first step to create an implementation model is the creation of a set of information requirements based on the implementation context. As described in Section 3.3, implementers, applications and application contexts are primary components of an implementation context, which can be regrouped and connected in Figure 3.14. As shown in this illustration, application users and implementers are considered as the most concerned stakeholders; implementers should develop integration solutions to integrate existing applications, probably by creating new applications to facilitate the integration; data sets and data schemas, either standardized or vendor-defined, are also components in concern to represent domain knowledge. The use of these contextual components (including domain knowledge) will be introduced to ensure the pragmatics of information requirements (Figure 3.15).
An important issue in STEP modeling architecture is that information requirements are not designed to be interpreted by computers, but rather by human beings. Information requirements are the main leverage to create (for model developers) and to use (for implementers) information models in a modeling architecture at a semantic level. Properly composed, information requirements are useful for both model developers and implementers. Model developers use them to generate implementation models. Implementers use them as a reference directly describing the implementation models. Hence, all semiotic aspects of information requirements should serve a purpose of effective human interpretation.

Syntactically, the information requirements should be expressed in a form familiar to implementers. UML (Unified Modeling Language) may today be the graphical modeling standard that is most familiar to implementers. Therefore, this study describes the information requirements in a form of simplified UML class diagrams (Rumbaugh, et al., 2004). The simplification is made to increase readability by omitting elements either obvious or redundant: Class compartments, indicators for abstract class, ease of navigation for aggregation, multiplicity and visibility. Figure 3.14 and Figure 3.15 exemplify this simplification style. This simplification does not violate completeness but increases readability. To sum up, the syntax is a graph with nodes and edges, decorated by arrows defined in UML. Note that in future there might be another solution replacing UML, and the encapsulated data schemas remains unchanged.
Semantic aspects of information requirements are limited, because computer interpretation is not required. Still, semantics of information requirements will greatly determine semantics of implementation models. Hence, the semantics of information requirements should be communicated at a suitable degree of granularity, completeness and validity. Thus, a controlled vocabulary is demanded. This mission is almost the same as creating an application ontology: Both tasks and domains should be depended on (Guarino, 1998). This study selects a series of principles (Table 3.4) for ontology design defined by Arp, et al. (2015), which are usable for representing information requirements.

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realism</td>
<td>Represent only those in reality.</td>
</tr>
<tr>
<td>Perspectivalism</td>
<td>Do not seek to represent entire reality. A modular approach is promoted.</td>
</tr>
<tr>
<td>Fallibilism</td>
<td>Change is possible, trackable and open to users.</td>
</tr>
<tr>
<td>Open-world</td>
<td>Allow continuous extension and amendment to adjust broadness and granularity.</td>
</tr>
<tr>
<td>Adequatism</td>
<td>Allow for multiple levels of granularity to adapt needs in multiple domains.</td>
</tr>
<tr>
<td>Relevance</td>
<td>Include terms used by relevant systems.</td>
</tr>
<tr>
<td>Consensus</td>
<td>Strive to ensure maximal consensus among stakeholders.</td>
</tr>
<tr>
<td>Overlap identification</td>
<td>For reuse between modules, identify synonyms, terms with different meanings, terms used at different levels of granularity.</td>
</tr>
<tr>
<td>Singularity</td>
<td>Use singular nouns for nodes.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Try to avoid uppercase and abbreviation.</td>
</tr>
<tr>
<td>Univocity</td>
<td>Hold exactly one meaning for each term (node) and each relation (edge).</td>
</tr>
<tr>
<td>Avoiding mass</td>
<td>Represent only countable terms.</td>
</tr>
<tr>
<td>Universalism</td>
<td>Represent only what is general.</td>
</tr>
</tbody>
</table>

Table 3.4 Principles usable for representing information requirements.
Pragmatically, information requirements should be effectively interpreted by both model developers and implementers. Information requirements from an implementation context is essentially a compromise among relevant contextual components: Knowledge of relevant domains, integrated applications and implementers (Figure 3.15). This compromise should be properly documented and tracked. 1) Domain knowledge is a critical pragmatic part of any implementation practice. The existing data schemas are considered as the most important source for the domain semantics, because the direct implementation goal is system integration conforming to the data schemas. Moreover, a formalized data schema is supposed to be highly reliable compared with knowledge of application users. 2) As shown in Figure 3.15, the information requirements should also address concerns of implementation. That is, information requirements should resolve foreseeable mismatches between data schemas and common applications. For instance, interfaces of applications may have different preferences on either global or local coordinates. 3) For implementers, the most important pragmatic issue is in relation to programming languages and should be directly reflected by realization techniques of the implementation models.

Figure 3.15 Information requirements based on implementation contexts.
3.4.4 Creation of an implementation model

As defined earlier, an implementation model is also an information model, with its specialized semiotics (Table 3.5). An implementation model is recommended to be realized in an API. Hence, syntax should be governed by specific programming languages. Semantics of implementation model is largely determined by information requirements. The mapping between the information requirements to an implementation model should be consistent and clear, i.e. each class and each relation in-between in the information requirements should have corresponding elements with (almost) same naming in the implementation models; thus, the information requirements in simplified UML class diagrams can be a document to brief the semantic structure of the API to implementers. Pragmatically, an implementation model should be interpreted by implementers with its API documents.

<table>
<thead>
<tr>
<th>Implementation model</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Pragmatics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Programming language, e.g. Java</td>
<td>Information requirements</td>
<td>API document</td>
</tr>
</tbody>
</table>

Table 3.5 Specifications facilitating semiotics of implementation models.

With semantics about application domains defined by information requirements, the remaining task it to implement the information requirements in a form of API with proper documentation. Thus, properties, methods and algorithms should be completed based on the information requirements. This is not a study focusing on software engineering; therefore, the art of designing a good API is not investigated in detail. However, there are still some noticeable fundamental principles (those principles explicitly regarding definition of semantics and syntax are excluded):

- Documentation matters. Any API is all about reuse; the reuse, even for components with best design, cannot happen without proper documentation (Parnas, 1994).
- Document before coding. Thus, documents are not affected by coding (Henning, 2009).
- Documents “read like a book”. Documentation should be kept as simple, intuitive and elegant as possible (Jacobson, et al., 2011).
“Smaller is better”. The fewer the elements, the easier it is to interpret, learn and use; adding any non-fundamental functions should be double checked (Henning, 2009). For an API in use, “you can always add, but you can never remove” (Bloch, 2006).

“Agility trumps completeness”. Start API with specification as short as possible, so that it is easy to read and maintain (Bloch, 2006).

Dictate semantics (Henning, 2009). What have been defined by information requirements should be obeyed throughout the entire content of the API.

Keep consistency. For such a complex (interdisciplinary) API, ergonomics depends on consistency which should always be maintained to ease use, memorizing and transference of learning (Henning, 2009).

Moderate impacts of implementation. As always, API is implementation-oriented, but cannot be over-specified (Bloch, 2006), i.e. impacted by implementation details too much.

Demonstration matters. A live demonstrable case can directly point to its value for users. Thereby, it is important to select significant use cases and working on sample codes for them (Jacobson, et al., 2011).

Note that, with the help of implementation models, implementers may still feel heavy loaded because knowledge about relevant domains seems too much. Figure 3.15 shows that implementers are supposed to possess knowledge in both domains, in software engineering as well as in production engineering. Generally, the assumption can be true because requirement engineering and system architecting help to bridge the domains. However, for professional system development, the requirement engineering and the system architecting can still be a huge workload. An implementation model can relax implementers from knowledge about information modeling, but cannot relax them from knowledge about the application domain. Chapter 4 aims at the latter relaxation.
3.4.5 An exemplification

In the appended paper D, the STEP toolbox API is an example to realize an implementation model. In this example, objectives are to simplify the traditional implementation process and to reduce learning effort of the implementers. This is exactly why an implementation model is incorporated. The implementation model is used to increases the pragmatic interoperability of an information model. Although not explicitly stated in the paper D, the STEP toolbox is developed based on the implementation context, so that its pragmatic interoperability can be ensured.

Kinematic data exchange is a target to be implemented with a module in the STEP Toolbox used in paper D. Implementers are supposed to use the STEP Toolbox API to operate data sets directly. Therefore, experienced model developers can begin the creation of the implementation models with analysis of exemplified instantiation. For instance, the following excerpt (Figure 3.16) represents description of a kinematic link and its relation with product definition. In paper D, there are also excerpts for a kinematic topologic structure and relation with geometric information.
A valid sample data set (e.g. Figure 3.16) may be a suitable start point for implementers to understand a standard, because this is an expected output, but it is not enough for developing an implementation model. An implementation model should support a reasonably complete scope for basic I/O operations. Its success is determined by success of implemented applications to interpret and process possible data sets. This interpretation process regarding data sets is the main focus of system integration and the reason why information models are needed. The implementation models should help implementers to use the data schemas rather than to process a limited set of data sets.
Based on the analysis of the data schema and other components in the implementation context, model developers can elicit information requirements, diagrammatically presented in Figure 3.17. The information requirements are presented here in a simplified form of the UML class diagram which is understandable by implementers. This illustration includes simplification that temporarily omits attributes and basic functions. As illustrated, nodes and edges are used to consolidate complex data structures in the data schemas. What has been hidden is difficult for implementers but easier for model developers. Hence, the complexity does not disappear, but is managed by more suitable experts.

The implementation models should reflect all the elements with additional consideration about the integrated systems. A major concern about the systems in this case is about translation of coordinate systems. In AP242, placements of joints are coded in links’ local coordinate systems, but these placement values may not be suitable for some applications only supporting a global manner for all inputs. In this example, not only is a general utility for coordinate system translation provided, coordinates for all kinematic elements in both manners are also provided explicitly. The result is a UML class diagram with more complete specification on properties (Figure 3.18).
3.5 Layered architectures

Section 3.1 and 3.2 form the gap of this study with identification of the root cause. The gap leads to the introduction of the implementation contexts (Section 3.3) and the implementation models (Section 3.4). Nevertheless, the previous sections focused on what these new concepts were and how to develop them, but how to use them was rarely reached. Hence, the introduction of a modeling architecture is needed to make use of the new concepts in real business circumstances. This study does not define a specific solution of architectural design for application software. Instead, a modeling architecture is put forth as an infrastructure to facilitate system integration implementation toward maximum pragmatic interoperability.

There are actually noticeable gaps in implementation practices, as reasons why a new modeling architecture is needed in an implementation context: 1) Standardized information models are partly incompatible with a context of use of application software, due to a limited set of information requirements. As a result, function design of some applications was limited
by demanded compliance with standards (Lee, 1999). This fact indirectly hurts the supposedly implementation independency of standards. 2) There is a lack of inter-protocol solution in existing modeling architectures. An explicit geometric context to increase semantic coherency is valuable for communicating design intent (Hedlind and Kjellberg, 2015). This can lead to heavy workloads for implementers using protocols without explicit geometric representation. 3) In manufacturing industry, there is essentially a lack of resources for an interdisciplinary errand like implementing information standards. When an information model is designed to be kept away from the implementation context, the learning curve to implement an unfamiliar information standards becomes steep.

Hence, a modeling architecture is needed to provide implementation-oriented support, taking advantage of the proposed implementation models. The core part of the modeling architecture is an implementation model as a contextual layer illustrated in Figure 3.19. It is designed for integrated applications (as in the top layer of Figure 3.19) interacting with data sets (the bottom layer of Figure 3.19). The composable layer is to take advantage of the original STEP modeling architecture that guarantees development of good information standards. The contextual layer, is to bring portability into the consideration to reduce complexity in implementation. As discussed in Section 2.3, the layered design was to compensate the traditional composable solution by a contextual solution that CAx system implementers are more familiar with.

Figure 3.19 The layered modeling architecture.
3.5.1 The composable layer

For the composable layer, generic data schemas are adopted which can be instantiated in specific application contexts, such as STEP AP242 (ISO, 2014b) or STEP AP214 (ISO, 2010a). The generic data schemas are adopted due to the comprehensive semantic representation and high extensibility that are exhibited in different aspects. 1) For starters, these data schemas are applicable with existing useful implementation methods, e.g. SDAI and its bindings to common programming languages. Apart from SDAI, definitions of validation methods and exchange structures (such as XML representation and binary representation) constitute solid foundation for implementation and use. 2) A generic data schema usually has a large scope with a regular incrementing plan to facilitate a complete support for a large range of industrial sectors. For example, STEP AP242, initiated as a convergent application protocol to harmonize AP214 and AP203, aims at satisfying interoperability requirements from automotive, aerospace and defense industry (ASD, 2009). 3) The comprehensive infrastructure and the large scope make generic information models easy to extend to new application domains. A plenty of examples can be observed during development and application projects. Chen, et al. (2011) confirmed that STEP AP214 was the only standard able to represent all needed modules for factory layout, although complexity and efficiency were still highlighted issues. Hedlind and Kjellberg (2014) proposed extensions based on STEP p105 ed2 (ISO, 2014a) to explicitly represent function requirements on kinematic errors, which could be easily integrated into AP242 as well. Lundgren, et al. (2016) proposed a necessary extension for STEP AP242 and STEP-NC (ISO 14649) to enable system integration for process planning and quality assurance.

With the generic data schemas as a prerequisite, the composable layer represents the default implementation framework specified by ISO 10303. This framework is built in a typical composable way but inadequate for implementers, of which the semiotic aspects was discussed in Section 3.4. The provided implementation methods have no clue about any semantics, i.e. all the useful utilities are at a syntactic level. Hence, to use the existing implementation methods, a large chunk of code is needed to populate the semantics. Moreover, even an executable program with SDAI cannot guarantee a valid data set without validation. Validity of a data set depends on
the instantiated data schema. Besides EXPRESS, SDAI and validation, there are many specific concepts regarding modeling that should be comprehended. Some of the concepts may have little connection with either the software engineering domain or the manufacturing engineering domain, which are only related to information modeling. As a result, this layer has low initial usability for novice developers or in early iterations, but has high final usability for experienced developers or, sometimes, in late iterations.

3.5.2 The contextual layer

The contextual layer, i.e. the implementation model (Section 3.4), consists of components highly related to implementation details. It is designed to be usable for implementers as the primary users of the information models. This layer is directly affected by an implementation context and, therefore, only works with corresponding implementation methods. For example, Java is chosen as the major programming language in paper D to implement the layer, according to technical requirements of related projects. In addition, the application context should be also reflected in this layer, since the implementation context is greatly dependent on the application context which is mostly concerned by implementers.

Programming interfaces, e.g. written in Java language, are composed and packaged directly related with domain semantic concepts and relevant functions. In this way, the contextual layer facilitates specific information requirements as well as functional requirements elicited from an implementation context. Moreover, references of the interfaces should be presented in an implementer-friendly way, i.e. adopting semantics widely accepted in the manufacturing industry and the IT industry.

As stated in Section 2.3, the contextual approach has several advantages which make it usable for the implementation context. 1) The contextual layer has a high findability, where specific components are easy to locate for specific requirements. The components greatly related with the contexts are easy to learn and fast to use by the implementers with little knowledge about information modeling or with limited sources to perform formal regular model mappings for implementation. 2) The contextual layer provides “out of the box” solutions for specific implementation contexts. The implementers are provided with efficient ways to realize semantic concepts with predefined default values. Commonly used data sets like
p21 data sets can be interpreted, created, or processed directly in an efficient but still valid way. The layer can even provide extremely contextual solutions that are only suitable for integration with a specific CAx system.

3) From guidance tutorials to sample codes, the contextual layer should deliver sufficient knowledge support for the implementers to implement frequently used processes. Most implementers involved in projects related to the study had limited knowledge in either specific information models or general information modeling principles. Skipping most parts of manual interpretation of the data schemas can benefit the implementation targeted on specific contextual needs with a high degree of efficiency and error prevention. 4) The contextual solution with high efficiency and high learnability in specific contexts will motivate the implementers, especially the end-user programmers, to use the model driven solution more often.

3.5.3 Closed layered and open layered architectures

The overall layered structure of the architecture implies dependency from top to bottom, which has two ways to apply the architecture for implementation: Closed layered and open layered (Matha, 2008).

Contextual use of the architecture is closed layered and one-directional (Figure 3.20), where the developed applications only interact with the contextual layer, but do not talk to any other layer; the contextual layer only interacts with the composable layer but does not have any idea about the application built upon it. The contextual use is suitable for novice implementers or implementation projects that can be satisfactorily supported by the contextual layer. Novice implementers do not have to interact with the composable layers, because closed dependency between layers is regulated. A fully closed way to use the architecture is a violation of the encapsulation principle, which may decrease reusability and maintainability of applications with dependencies of more than one layer (Martin, 1996).
On the other hand, composable use of the architecture is one-directional but open layered (Figure 3.21), where the developed application can interact with both the contextual layer and the composable layer. That is, the contextual layer is “opened”. The composable use is suitable for expert implementers or implementation projects that cannot be fully supported solely by the contextual layer. This open channel from the contextual layer to the composable layer also helps novice implementers understand the composable layer as a generic solution based on a generic application context, which makes the learning process more efficient.

Hence, the architecture should provide valid results with each of the single layer or with a combination of both layers. Expert implementers should be allowed to switch between both layers freely even for one data set. The open layered architecture should also be open enough so that the implementers can reach composable components beneath any contextual components for deep understanding of the implementation model.
This layered design aims at a portable solution for model driven system integration, with reduced complexity, increased efficiency and better learnability, compared with the conventional composable approach. A complete solution for all contexts is not the goal of the architecture. After all, aiming at all contexts means no concerned context at all. Instead, this study delivers architecting principles to make information models efficiently used in the implementation context with the aim at full interoperability. During the course of this study, this architecture has been refined, used and reused internally and externally in several projects. Concepts and functions are shared among the projects with similar or overlapped application contexts, so that contextual components at a proper level of abstraction built for one project can be efficiently reused by another project. In this way, the architecture, as a methodologic concept, helps engineers to reuse previous knowledge for problem solving (Spinellis and Gousios, 2009).

3.6 Discussions

The proposed modeling architecture can be seen as an enabling system (ISO/IEC/IEEE, 2015) to serve system integration using information models. It benefits the implementation context in at least three aspects, i.e. a high level of architectural agility, a high level of component decoupling and standardization of contractors.

The engineering context is essentially a changing environment and system integration is an important technique to cope with changes. The challenges are faced by IT developers as well as IT users. For long-time application in use, many factors of computerized information systems are constantly changing. From a technological perspective, hardware platforms, software frameworks and engineering systems are likely to be outdated or replaced at any time; implementers often need to switch programming languages or patterns to cope with the technical changes accordingly. From a business perspective, fast time to market (TTM) is an eternal demand, which puts engineering tools under continuous review for changing or upgrading; besides, potential changes on computerized information systems may also be triggered by new market demands, new regulations and new enterprise structures.
The changing environment demands an ability to make rapid responses by involved systems, i.e. high agility. The proposed modeling architecture can support effective and efficient implementation by reusable contextual solutions complementing the extensible composable information models. Thus, system integration can be implemented in accordance with unpredictable changes in an agile manner. Prototyping is easier to be implemented with such an agile infrastructure as the new modeling architecture.

System integration is always an ongoing project with possibilities of integrating new systems and disconnecting existing systems constantly. This demands ease to couple and decouple systems. Moreover, just like common implementation projects, dependency reduction is a core idea to achieve a high agility. The modeling architecture provides a solid basis for the dependency reduction. As illustrated in the upper scenario of Figure 3.22, with four exemplified systems communicating with each other, any change made in one of the four systems may cause at most three changes of communication mechanisms; a new additional system needs to establish four communication mechanisms in order to be integrated. An independent contractor reduces the dependency of integrated systems, as a leverage to replace the tedious communication mechanisms (the lower scenario of Figure 3.22). Information models acting as the contractor encapsulate all interfaces. Hence, only one change of integration topology is needed for coupling and decoupling of any system. The new topology isolates each system and enables each system to evolve independently, which greatly reduces coordination effort and increases implementation efficiency.
There are also potential downsides of introducing a contractor, e.g. increased complexity as a new component, decreased system performance with one more step for communication and a huge independency issue if the contractor is changed. Even though all the downsides are potential, these downsides contribute to a large part of a trade-off when developers need to decide whether to introduce a contractor and how to choose a suitable type of the contractor.

Standardization, as a major leverage of the proposed modeling architecture, is also a major feature of the contractors to deal with the above mentioned downsides. A well accepted standard often implies that experienced developers have already taken implementation complexity and performance into consideration. A mature standard suggests low changing possibility or at least backward compatibility. Besides, when using standards, there is usually a large resource pool for teambuilding and consulting and a better chance to reduce workloads for integration with other systems (which may be compatible with the same standard). Leveraging standards helps developers to adopt best practices instantly and to gain a better chance to be understood externally with a common framework and terminology. To sum up, standardization provides a usable, stable, and reliable infrastructure to optimize design for system integration.
Chapter 4
A model driven implementation process

*Design is not just what it looks like and feels like. Design is how it works.*
*Steve Jobs*

This chapter is about how to use a developed standardized generic information models. A magnificent standardization effort in production engineering is ISO 10303 STEP and the related standards (more details in Section 2.2.2). Modeling architectures of these standards are major references to discuss behaviors and structures of information models in this chapter.

These standardized information models are commonly treated as a technical constraint (Section 4.1.1) during implementation. However, their usages could be beyond this role (Section 4.1.2), because they are also notable for 1) focused concern with high semantic interoperability and 2) being used with a procedure similar to conventional model based software implementation processes. Hence, the standardized generic information models are potential to underpin the software implementation process (Section 4.1.3).

To fully take advantage of this potential, it is necessary to understand 1) what activities of the software implementation processes can be supported (Section 4.2), 2) what deliveries of the activities correspond to the information models in sequence or abstraction levels (Section 4.3) and 3) how to use the information models within the activities of the software implementation process (Section 4.4). The activities are grouped into five stages (in Section 4.5 through Section 4.9) and are underpinned by artifacts of the modeling architectures. The activities focus on delivering architectural design decisions to help industrial practitioners to rapidly generate feasible system integration solutions.
4.1 How to use information models

Today, information models are used in a limited way in practice in software engineering. Toward system integration for product realization, implementation is characterized by rich content and intensive human-model interaction; it is inevitable to use standardized generic information models to facilitate high interoperability in this context.

4.1.1 Current limited use of information models

To use an information model, typically implementers necessarily follow a manual mapping process (Figure 4.1), to understand and integrate a data schema. In this mapping, activity models (e.g. AAM in ISO 10303) and reference models (e.g. ARM in ISO 10303) are intermediate models to bridge interoperation needs in reality with a computer interpretable data schema (e.g. AIM in ISO 10303). This mapping process is defined in modeling architectures aiming at facilitating valid use of the information standards (more details in Section 2.2.2). Without the mapping process, it would be frustrating for anyone to locate right parts of a data schema to represent what should be communicated in a specific application context.

![Figure 4.1 A traditional mapping process for using a data schema.](image)

If strictly obeying a modeling architecture (e.g. for ISO 10303), manually following this mapping can be a heavy burden on implementers. This fact violates a general intention of standardization, i.e. to reduce implementation burden (Lewis, et al., 2008). Particularly, implementers need to interpret the intermediate models such as activity models and reference models (Figure 4.1). As a result, modeling architectures are suitable for semantic interoperability and computer interpretation, but not suitable for pragmatic interoperability and human interpretation.

Now, zoom out to see the bigger picture, i.e. implementing system integration in an engineering context, where the use of the standards is a
portion of implementers’ work: They map needs for communication (i.e. information requirements) to computer-interpretable data schemas (the middle column in Figure 4.2). Besides using the data schemas, implementers map other needs (e.g. independent content, independent functions and qualities) to an application. The mapping from reality to application is the mainstream in implementation (the left column in Figure 4.2), even without information models.

![Figure 4.2 Current integration of the information model mapping with the implementation process.](image)

These two processes, the information model mapping and the implementation process, are usually independent until the data schemas have been delivered as a technical constraint of construction. It means data schemas are commonly used as a standard to strictly follow. The intention of this use of standardization is right, i.e. reuse of existing product information and generating reusable product information. At most time, implementer designers select standards in early stages, e.g. requirement analysis,
but use them only in late stages, e.g. programming and testing. This situation, using standards as technical constraints in late stages, is observable when aiming at syntactical interoperability, e.g. using HTML, IMAP and TCP/IP. The standards regulate syntax for integration, which sounds familiar for implementers.

4.1.2 Envisioned use of information models

However, the current limited use of information models (Section 4.1.1) is a waste of talent and knowledge. Beyond regulating syntax, the standardized information models (e.g. ISO 10303) are characterized by the focus on interoperability for rich semantics. All information captured in the information models should be formally self-descriptive, to guarantee full semantic interoperability and computer interpretation. Usually, this is creation of top experts in subject domains. The knowledge within the information models is potentially highly useful for almost all stages of an implementation process, rather than just the construction stage.

In particular, system integration in this subject domain is challenged by integrating information architecture with software architecture. The involved context could be expanded to a large business domain (design, manufacturing, and resources) and a large technical domain (client application and server applications), as stated by Rosén (2010). This integration is characterized by complicated content-rich information flows and processes. This context has been formally represented by many artifacts in modeling architectures, such as the activity models and the reference models.

To sum up, information models possess some unique features in the standardization world, i.e. rich semantics on domain knowledge, creation of top domain experts, used in content-rich system integration, and needs to support computer interpretation at a semantic level. These facts lead to troubles and opportunities when using an information model. Both Section 3.2 and Section 4.1.1 discussed the troubles. To solve the troubles, Chapter 3 changed modeling architectures to support implementation; this chapter changes the way to use information models during implementation.

To achieve efficient and successful implementation, this study presents an enhanced implementation process closely integrating the manual interpretation of the model artifacts (Figure 4.3). In the enhanced implementation process, the standardized information models can potentially
support implementers in forms of comprehensive domain-specific knowledge. Meanwhile, it is possible to ease the manual mapping with early awareness of the standards in an implementation process. For instance, the activity models and the reference models can greatly benefit user study and requirement analysis. Consequently, the information models become support of implementation rather than just as technical constraints.

Figure 4.3 Closely integrating the mapping process with the implementation process.

4.1.3 Development of an integrated process for the envisioned use of information models

Implementation traditionally progress within a structured process with stages. Where, when and how to integrate the information models with the progression should be specified. This is crucial since most developers have limited knowledge and skills in information models, subject domains, or
software engineering. The following parts will reason how to use the information models in the implementation process with stages.

Essentially, the implementation process is also a mapping process from needs to accomplish goals or experiences in reality to a system satisfying this reality, with a process from context research to construction. This process is similar to the model mapping process. Thus, it will be helpful to examine the mapping processes in different domains and to discern similar stages that can support each other. This examination can start with a generalized mapping pattern faced by today’s engineers, i.e. from complex concrete entities in reality to another type of complex concrete entities in reality (Figure 4.4) which are realized products (including digital products). For instance, software engineers translate users’ needs to developed application software; manufacturing engineering translate business needs and customers’ needs to physical products during a product realization process.

Models are adopted as an abstraction of concerned facts to translate the reality to the products (Figure 4.5). By definition, a model is usually an abstraction of an existing system or a future system. In the domain of engineering, a model is an abstraction to document a system design (Reeves, 2005). Engineers live with the models to design and communicate without considering irrelevant details in every corner of a context of use. In manufacturing industry, models describing functions and forms are used to translate needs to physical products. In IT industry, models such as UML (Unified Modeling Language)-based diagrams or source codes are used to map user requirements to compiled programs executable in run time. The mapping will exist for ever, as long as human beings cannot find a better mechanism to directly translate the reality to the finished products.
The term “model” has different definitions in different engineering fields, but it is common that models are used to represent, to communicate, or to simulate what is designed. Models play a critical role in the scope of this study (Figure 4.6): In this scope, implementers use the system design models (e.g. UML or HCD models) to describe functions and content at different levels of abstraction to be delivered in software; production engineers use engineering design models to describe what should be manufactured. Hence, modeling activities universally translate user needs and business needs in a context of use into a product.

![Diagram](https://example.com/diagram.png)

**Figure 4.6** Software implementation supports (physical) product realization in manufacturing industry: The scope of this chapter.

Particularly, applications targeted on system integration need the engineering design models (the cross point in Figure 4.6) facilitating functions of interoperability: Communication, interpretation and processing. Hence, the engineering design models should instantiate standardized data schemas (upper left of Figure 4.7). This is where information model mapping is involved. The two processes, information model mapping and software implementation, are performed by implementers. Both processes share the same input and output, i.e. application contexts and application software.
Moreover, between the input and output, intermediate models are involved in both processes to describe what should be implemented. The intermediate models compensate the left part of Figure 4.7 for information model mapping, which becomes Figure 4.8. In this new illustration, the added activity models and reference models formulate the problems described in Section 4.1.1. Implementers should perform the two parallel processes (Figure 4.8) which are separated by shared similarities in sequence and abstraction manners. The similarities imply opportunities to save implementers’ workloads by reusing the existing model artifacts during some implementation stages (Figure 4.9).
4.2 The essence of software implementation activities

Now that the implementation need to be underpinned based on its stages, it is necessary to clarify what stages should be underpinned. Existing studies of engineering activities in general reveal basic characteristics of implementation processes performed by human engineers, in terms of actions, sequences and iterations. In the engineering context, design is a core activity and an encompassing concept in any form of implementation that performed by engineers.

It has concluded that there was no structured sequential process for engineering activities especially in a complex context (Fricke, 1996), except a common categorization of the activities being performed flexibly and iteratively: Analysis, synthesis and evaluation (McNeill, et al., 1998, Figure 4.10). This finding coincided with a triple-mode pattern that led to novel design decisions: Examination, drawing and thinking (Akin and Lin, 1995). Nevertheless, sequential definitions of the design processes in a macro scale were still recommended and generally observed. For instance, Akin and Lin (1995) made segmentation based on time stamps of protocols; they studied and concluded three segments: Problem understanding, design and retrospection, where the design segment was divided into three phases: Conception, development and representation (Figure 4.11). “Design” was used only for the second segment, and problem understanding and retrospection were defined as separated parts of the model. However, it is hard to say that designers need not understand problems before drawings and do not review their work from time to time.
4.2.1 Stages of software implementation

The triple-mode pattern is universally applicable not only for engineering design activities, but also for software implementation activities. This applicability can be confirmed by widely-accepted modern software implementation styles where extensions may be made to the pattern according to specific contexts. Three styles are elementary to set the stages of all modern software implementation methodologies: 1) The waterfall is useful to understand the basic stages and the necessity to avoid iteration; 2) the incremental style and 3) the prototyping style focus on right ways of iteration.

A waterfall-styled process (Royce, 1970, Figure 4.12) was a common traditional way to describe how to ideally develop software, also based on a target of bridging the reality with produced programs (Figure 4.4). This waterfall style is useful for sequential introduction of important stages in an implementation process. The proposed process did not deny the existence of iterations, but emphasized using complete in-time documentation and stakeholder involvement to eliminate unwanted iterations. Note that,
the referenced “design” activity in Figure 4.12 was defined not as an encompassing concept, but as a limited concept to simulate and plan the coding and the testing.

Figure 4.12 The waterfall software implementation style aimed at eliminating unwanted iterations (Royce, 1970).

However, software implementation has never been ideal. It is hardly possible to specify all functions and content, as well as complete definition of forms and behaviors of system components, at an early stage. It is always valuable to bring back the iterative fashion (Figure 4.10), even at a macro level. Hence, iterations are often inevitable and can be helpful if managed properly. There are two basic styles of iterations depending on deliveries of each iteration: Increments or prototypes. Both styles deal with limited resources and changing contexts by iterating the waterfall style to a certain extent (Figure 4.13). Reflecting analysis of results, refining products and evaluating with stakeholders, both styles practice the analysis-synthesis-evaluation pattern. These two elementary styles are the basis of almost all modern software implementation approaches.
The incremental development style emphasizes design decisions at two levels: Overall design for high-level structure of all possible increments of a system and detailed design of each increment. Iteration of the incremental may begin with one of the three different phases: Requirement specification, overall design and detailed design (CMS, 2008), but the detailed design is mostly often revisited for a new increment. The delivery of each iteration should be an operational system with all developed increments merged. The incremental strategy has been widely adapted in different development methodologies. A typical example is the Unified Process (UP) with its most famous variation, the Rational Unified Process (RUP, Kruchten, 2003).
On the other hand, the prototyping style uses prototypes as a fast-developed simplified abstraction for testing to help developers refine design decisions until a design is feasible enough for formal implementation (Bally, et al., 1977). Stakeholders are recommended to be constantly involved from requirement specification to testing, especially in refining iterations of the prototypes. The prototyping strategy has also been integrated within many popular development methodologies and possibly applied together with the incremental strategy.

4.2.2 Common software implementation approaches
The two strategies with emphasis on fast and lightweight iteration set a conceptual foundation of most modern software development frameworks which compensate strategies with organizational principles at different detailing levels. At a low detailing level, agile software development methodology presents principles widely accepted in today's IT industry: High adaptability, less relying on prediction, continuous delivered products, fast responding to requirement changes, more individual communication & less documentation, fast iteration cycle time, etc. More or less, these principles can be found in established development frameworks at a high detailing level.

Extreme programming (XP) and Scrum are two well-known frameworks to apply the agile development process at a high detailing level, which are recommended to be fully integrated in a single project. XP tends to depict practices that advocate principles of the agile development methodology and Scrum focuses on regulated organizational processes with designated roles. Both frameworks value rapid iteration but XP may have shorter cycle time (1 to 2 weeks vs. 2 weeks to 2 months). The implementation sequence of features can be adjusted in XP, but not adjustable in a “sprint” of Scrum. However, priority of features determines the implementation sequence of XP but does not affect the implementation sequence within a “backlog” of Scrum. What's more, principles of XP contribute to many other types of engineering practices as additions or adjustments of the agile development methodology e.g. continuous integration, pair programming and refactoring. Scrum does not use the practices to regulate how the framework to be applied in real work. It is possible for Scrum projects to integrate the engineering practices defined in XP.
Large software vendors also propose specific development processes that may be well suitable in some context and reflect the three-mode design activity pattern (analysis-synthesis-evaluation). Microsoft proposed a process called “synch and stabilize” when developing Windows 95 (Cusumano and Selby, 1997, Figure 4.14) to specialize the incremental strategy. The process was very suitable for a “small team” culture of Microsoft and should be effectively supported by usability labs through all the three phases. The process was iterated toward milestones defined by targeted features and targeted features were typically divided into three parts for increments according to pre-defined priorities. Nevertheless, the process served as an organizational strategy rather than a technical strategy. It did not specify modeling details to document requirements and to communicate solutions and designs. The roles of designers and programmers were blurred in the second phase because both design and coding were suggested to be performed by so-called “developers”.

Figure 4.14 Microsoft “Synch and stabilize” process (Cusumano and Selby, 1997).

4.3 Model based software implementation

The activities of software implementation rely on models as deliveries. System design modeling (as shown in Figure 4.6) is an important methodology to convey the ideas, reasoning and decisions of a design in software implementation. An important modeling principle is architecting based on viewpoints, driven by use cases or scenarios (ISO/IEC/IEEE, 2011b; Rozanski and Woods, 2012). For example, designated viewpoints are the basis of the 4+1 view model (Figure 4.15. Kruchten, 1995) to generate and document software architecture. Some of UML modeling techniques were selected to
develop and describe the software architectures, e.g. class diagrams, use cases, interaction diagrams and packaging diagrams. These viewpoints were supposed to feed different stakeholders of a product: Project managers, system engineers, programmers, clients, testers, etc., so that each of them could conveniently view a specific aspect of an architecture design. For instance, software implementers need the logic views to represent business logics and domain concepts based on scenarios and functional requirements; the logic view is highly similar to the information requirements (Section 3.4.3) represented in the ISO 10303 modeling architectures.

The 4+1 view model demonstrated a typical viewpoint-based use-case-driven architecting process for designing software systems with probably a large amount of interaction. However, this design style has been raising criticisms for several reasons. The use cases, as a descriptive method to generate much information important for the entire development, usually left too many implicit assumptions of the system design (Constantine and Lockwood, 1999). Tasks were organized in a hierarchy, but no priority had been assigned to the tasks. There was also a lack of description of systems’ precise behaviors, which resulted in low efficiency and low flexibility during the development process (Cooper et al., 2014).

4.3.1 Involvement of the HCD paradigm
Such criticisms lead to new ways of architecting software concentrated on effectively modeling user needs in specific context of use, i.e. the HCD (Human-Centred Design) paradigm. In addition, the HCD paradigm meets the
needs of the CAx systems characterized with intensive human-computer interaction in the engineering context. Development of such systems demands effective information architectures for rich content presentation and manipulation.

Many HCD methodologies adopt the stages set by the waterfall style as an initial start point, with some extensions and iteration points. Sharps, et al. (2007) exemplified this style with a typical four-stage process (Figure 4.16). The setting of the sequence could be more flexible, which was argued by the influential “star model” (Hix and Hartson, 1993, Figure 4.17). It revealed high flexibility of the design process where designers should be free to connect the activities in any sequence if only through evaluation, rather than stuck to sequential or iterative rules. On the other hand, its strength also became its weakness in real use, because it made a project unmanageable, where no specific hint about progress was provided. Perhaps this was the reason why people reached a consensus of specifically defined interdependences of HCD (Human-Centered Design) activities (ISO, 2010b, Figure 4.18). In this standardized HCD process, the activity of evaluation was still a major trigger of iteration, but other activities became sequentially interdependent.

Figure 4.16 Four activities of interaction design (Sharps, et al., 2007).
Note that the mentioned processes mainly focused on the design activities isolated from a bigger picture, i.e. the technical process of application software lifecycle (ISO/IEC, 2008). To accomplish a final product delivering desired user experiences, cross-disciplinary effort has to be required. Thereby, Mayhew (1999) made an exceptional summary to integrate the HCD process with software engineering activities, specifically fitting the designers’ activities into the process of OOSE (Objective-Oriented Software Engineering, Jacobson, 1992). The result was called usability engineering lifecycle (Figure 4.19) which focused on modeling of every contextual element (e.g. users, domains, tasks and interactions) before detailed UI design.
Figure 4.19 Usability engineering lifecycle (Mayhew, 1999).
To sum up, models have been used in many approaches for software implementation, especially when integrating the HCD principles. There are pros and cons about the HCD. The HCD aims at avoiding too many iterations by accurate specification of user requirements and comprehensive understanding of content. That said, the HCD-based concerns are limited when it comes to either software architecting or software engineering, which may draw little attention of implementers. Namely, integrating the HCD principles exhibits more contextual modeling but less focus on technical design modeling. A generalized process can be derived based on studies on previous important implementation processes (Figure 4.20) for which the iterative fashion is encouraged with evaluation in each step.

4.3.2 In-depth participation of domain experts

What is more, the CAx systems are often too content-rich to achieve best UX (user experience) only by practicing design principles in software engineering and HCD. When the rich-content domain knowledge forms the bottleneck of the implementation, it is relatively easy for domain experts to dominate development of system architectures. Therefore, involvement of end users (i.e. domain experts) are highly recommended to ensure professional integration of domain knowledge. Based on observations, many system integration projects in industry to solve interoperability issues have been (partly, at least) driven by primary users (end-users or operators) of the systems. The implementation cases in the appended paper B, C and D are all developed closely involving stakeholders from the user side. This fact should be taken advantage of in an integrated implementation process.
Therefore, the result of this study is prepared mainly for the domain experts with little experiences or knowledge in either software engineering or information models. It enables delivering feasible design solutions ready for professional implementation (demonstrated in Section 5.2 and 5.3). Hence, this study focuses on generate architectural decisions related to domain knowledge. In general, deliveries of different activities in an implementation process are essentially sets of decisions with descriptions at different levels of abstraction. In this study, the implementation activities are recommended to be executed at a low detailing level, i.e. an architectural level (more details on system architectures in section 2.4). Thus, practitioners can agilely focus on decision-making to satisfy high-level implementation goals. Moreover, iterations with prototyping or increments can be analyzed, synthesized and evaluated in an agile fashion.

4.4 Process integration

The design principles and focuses are different between the software engineering perspective and the HCD perspective, which forms a gap between the software engineering methodologies and the HCD methodologies. There is another inevitable gap created when implementing with the information models, where implementers could expect heavy workloads (Section 4.1.1).

As discussed in Section 4.1.2 and 4.1.3, the potential of information models (i.e. rich semantics and procedural similarity) makes the information models useful for the entire software implementation process. Thereby, when implementing with information models, it could be valuable and promising to investigate the fitness of integrating the model mapping process with the established workflows of software engineering and HCD (Figure 4.21).

Figure 4.21 Processes need to be integrated.
The intermediate models (activity models and reference models) provides different levels of abstraction that is practically useful for application design. There are usually a hierarchy of multiple artifacts at different levels of abstraction to construct an activity model or a reference model. Hence, specific parts of these models can be related with specific implementation stages, according to the levels of abstraction. This relationship can be illustrated in Figure 4.22. This illustration shows that the existing artifacts of modeling architectures, as references, support different stages of the general stages of the conventional software implementation workflow. Note that this illustration implies the order of timing and abstraction levels only between adjacent entities linked by solid arrows.

![Diagram](image.png)

**Figure 4.22** The information model mapping process underpin a process to implement software at different levels of abstraction.

This supportive relationship in accordance with sequence and abstraction levels is the basis of integration of the different processes. The following Figure 4.23 is a brief illustration of this process integration in two dimensions. Horizontally, the general software development process is extracted from Figure 4.20. This process, to a certain extent, fit with both the
software engineering process and the IxD process which pay different levels of attention in different stages. Vertically, the information model mapping process is deployed for suitable stages of the implementation. Integration of the two dimensions creates a model driven software implementation process for system integration.

Figure 4.23 Integration of the processes.

Perhaps it is too conceptual to understand the process only from a two-dimension view, as there should be more detailed reasoning on how this integration actually works. This process can be detailed and illustrated as in the following Figure 4.24. Each stage is dependent on the previous one and there are two or three key activities for each stage. As recommended by trendy methodology of agile development (Brhel, et al., 2015), an iterative and incremental fashion based on timely evaluation should be applied for this lifecycle, rather than the implied waterfall style. No decision should be set in stone for each stage, otherwise the project may suffer from low flexibility in a changeable application context.
As mentioned in Section 4.3.2, this study is mostly concerned with interests of domain experts as key implementers for system integration. Although the process can be traced down to a construction stage, this thesis focuses on delivering architectural design decisions. That is, deliveries in most activities are descriptive artifacts at a low detailing level, i.e. contextual study for scoping, conception based on a narrative description, concrete design framework, a selected set of UML-based descriptive models and a set of lightweight wireframes. Therefore, even industrial practitioners or end-user developers with little experiences in software implementation are able to perform this process and deliver feasible solutions ready for professional construction.

In terms of sequence and abstraction levels, the integration at the level of activities can be illustrated in Figure 4.25 together with the traditional processes of HCD and software engineering. Most of the initial activities to analyze contexts and requirements can be supported by interpretation of the information models so that the effort of the model mapping is not only liable to the implementation. This integration can solve a key problem of the modern agile fashion to develop software, i.e. a lack of reliable domain-specific models in early stages or early iterations. The information models can be a critical support as a domain knowledge base for the developed applications to guarantee comprehensive data models and process models.
The integration is also a bridge for the separated activity concepts in the HCD and the software engineering. The HCD process is expected to be practiced in any type of development processes, but software engineers are used to focus on rapidly delivering viable products instead of acceptable user experiences. After all, not all systems need to be interactive. However, most system integration in the manufacturing industry is all about efficient interaction with engineers in the subject domains and HCD principles will be highly useful to deliver valuable applications. The rest part of this chapter will introduce each stage of this process with examples and references to appended publications.
A MODEL DRIVEN IMPLEMENTATION PROCESS

Figure 4.25 Activities of Model driven software implementation process for system integration and the parallel processes.
4.5 The scope stage

At first, scope of the application should be defined, which is also a key function of activity models describing processes from the perspective of the subject domains. Activity models provide first-hand informative support for goal identification and most of behavior patterns to study users. For instance, the ISO 14649 STEP-NC (ISO, 2003b) provides activity models generally describing (not prescribing) typical understanding of the process from design to manufacturing of products represented in IDEF-0 diagrams. The highest level of these diagrams is shown in Figure 4.26 which can be decomposed into lower levels of diagrams. The diagrams display domain specific information on data flows, functionalities and contexts.

![Diagram](image)

Figure 4.26 The highest level of the activity models defined in ISO 14649.

Designers are supposed to identify and select a part of the activity models that are useful for the targeted direct users of developed applications and to define the scope, based on the activity models in higher levels and other available resources, e.g. user research, market research and literatures. The appended paper F is an example of an extensive literature review on the scope and the context of engineering applications in manufacturing industry. Mature standards such as STEP-NC are often developed by industrial practitioners who are familiar with the application contexts. Therefore, the information models at this level can be greatly useful when there is a lack of resources to perform a comprehensive contextual study.
4.5.1 User studies

Software engineers traditionally consider their work starting with requirement analysis, but this convention is criticized by experts in IxD (Interaction Design), IA (Information Architecture) and UX (User Experience). Research on users’ needs and goals is a preferable initiation for most interactive system design. Modeling users to describe personal information of the target users is useful when system objectives are unclear before defining any requirements.

Persona (Figure 4.27 is a simplified example) is one the most common time-tested way to study and model the users. It is a synthesized depiction of targeted typical real users to study goals, behaviors and attitudes of the users. Abstraction and composite of these factors formulate the personas which assist the decision, design and evaluation of products.

**USER STUDY - A PROTO-PERSONA**

Brandon

“People forget how fast you did a job – but they remember how well you did it”

**AT A GLANCE**

AGE – 40
LOCATION – Stockholm
JOB – Senior process planner

**BEHAVIORS**

ANALYZING PROCESS PLAN INFO – Understanding process plan info associated with product design info, and identifying potential risks.
SYNTHESIZING RISK MODELS – Establishing risk interdependency, locating critical potential problems, and addressing countermeasures.
EVALUATING RESULTS & UPDATES – Being aware of changes in all models, influences of the changes, and completion states of the countermeasures; and adjusting the risk models.
EXPERIENCE NEEDS
ATTENTION - Staying focus on most critical issues
CERTAINTY - Feeling reassurance about all the issues
REUSE - Avoiding "reinventing the wheels"  INTEGRATION - Info in an integrated, controllable, tangible manner.
FLEXIBILITY - allowing a non-sequential, back and forth freely, iterative process
LIFE MOTIVATORS
BE PRODUCTIVE – Brandon is proud of his productivity and punctuality as is also part of organizational culture
FEELING ORGANIZED – Quality assurance is a critical work, but not his major work. He needs to switch easily in and out with this temporary team.
LEARNING MORE – Not just for expertise in process planning, he’d like to learn more about quality assurance, IT tools, product design, and line design.

Figure 4.27 An example of proto-persona for a user of risk assessment applications.

It is often unlikely to perform exhaustive classic user research obtaining detailed qualitative data for urgent CAx-related implementation in industrial reality. Hence, proto-personas as a provisional version of formal
personas are recommended as a tangible user model that less resources are required (Buley, L. 2013). A proto-persona usually explains users in a less narrative and more structured way than traditional personas. A proto-per-
sona (exemplified in Figure 4.27) as a tool is not an accurate method to generate and validate a design, but it helps designers focus their priorities on critical needs, critical tasks and critical workflows. Also, it provides a concerted empathetic view for team discussion during the implementation process.

For a user study, goals are drivers of behavior patterns that model the users. Individual goals are actually the ultimate concern of UX studies (Bevan, et al., 2015). In the context of personas, modeling the users is essentially descriptions of goals at different levels. It is always important to remember that the design of interactive systems should be goal-directed, where goals (why users act) motivate behaviors (how users act). After all, it is possible that users may act ineffectively or inefficiently in their daily lives, but their goals are correct. Hence, the user models should elicit the true motivations at first. Researchers may not be able to identify the user goals immediately. Sometimes, designers can understand the application domain so well that goals can be directly identified, but in most of cases it is a concentrated study on users that is needed to generate a comprehensive view of user goals.

Norman (2005) proposed three levels of cognitive and emotional pro-
cessing: Visceral, behavioral and reflective. They can be mapped to three levels of user goals: Experience goals, end goals, & life goals (Cooper et al., 2014):

- **Experience goals** define how users want to feel: Comfortable, pleased and enjoyed, although it is difficult to talk about the experiences “in the context of impersonal business”.
- **End goals** motivate performing tasks. Users’ “behaviors, implicit assumptions, and mental models” need to be complemented by product behaviors or product functions through tasks.
- **Life goals** represent personal aspiration beyond the subject con-
text and explain why end goals need to be accomplished.

End goals are considered as a core part that is concerned mostly by designers, especially in the context of business application development, so in proto-persona the end goals with related behaviors are addressed at
first. The three levels of goals are titled in an easier-to-understand way, i.e. behaviors, experience needs and life motivators.

In the appended publications for kinematic data exchange (paper A and paper E), the needs are derived directly from a local automotive producer. In the beginning, system scope looks confusing until the users’ needs are clarified (see Figure 4.28). Users for cutting tool data exchange, with classification, GD&T (Geometric Dimensioning and Tolerancing) and catalogues (paper B and paper C), share a similar model with kinematic data exchange. They are both expertise on process planning and manufacturing resource (e.g. machine tools and cutting tools) management and both play a similar role for their daily tasks.

**USER STUDY - A PROTO-PERSONA**

**BEHAVIORS**
- **ANALYZING AVAILABLE INFO** – Understanding product design info, with other info regarding manufacturing resources such as machining tools and factory layout design.
- **SYNTHESIZING MODELS** – Proposing process plans synthesized with available information.
- **EVALUATING PROCESS PLANS** – Checking the process plans along with production line models.

**EXPERIENCE NEEDS**
- **CLARITY** – Feeling clear about all the information.
- **CERTAINTY** – Feeling reassurance about all issues and no missing information.
- **REUSE** – Avoiding “reinventing the wheels”.
- **INTEGRATION** – Info in an integrated, controllable, tangible manner.

**LIFE MOTIVATORS**
- **BE PRODUCTIVE** – Scott tries to keep his productivity as required by the fast-changing industry.
- **BE RESPONSIBLE** – Although quality assurance is not his major task, he’d like to check his design as throughout as possible before handing to others.

![Figure 4.28 The user model for users of kinematic data exchange.](image)

### 4.5.2 Visions

User models describe the motivators and behaviors from the standpoint of users. Now it’s time to address a vision from the user models to satisfy user goals at a high level. Goals and, perhaps, frustrations have been highlighted, so that consensus about priorities can be established. Defining a vision essentially includes two steps both in a high level, i.e. a problem statement and a vision statement. The problem statement focuses on current frustra-
tion with a situation that needs to be changed and a cause-and-effect relationship between the frustration and the user goals. The vision statement addresses design objectives with a feasible answer to that problem by the provided product. Take the development of a system for risk assessment of process plans for example:

**PROBLEM STATEMENT**
Currently PFMEA (Process Failure Mode and Effects Analysis)-based risk analysis is performed in an unsatisfactory manner. There is a lack of systems to support necessary tasks to guarantee accurate and efficient quality assurance. The tasks include information integration, contextual analysis, problem identification, solution proposal, activity tracking and update management.

**VISION STATEMENT**
A model driven system design for PFMEA-based risk analysis will help users achieve accurate and efficient quality assurance. Highly integrated models are demanded to facilitate a holistic GPS (Geometrical Product Specifications)-facilitated view and a flexible interaction manner. Issues regarding fragmented data, untraceable workflows and stubborn spreadsheet-based interaction should be avoided.

The key part regarding information model is the information integration in the problem statement and the fragmented data in the vision statement, both of which reflect the large number of information sources for process planning displayed in the high-level activity model (Figure 4.26). Thus, prioritized expectations can be stated easily based on the problem and the vision. In the case of risk assessment, the prioritized expectation can be stated as: 1) A holistic view based on 2) a fully integrated model with all pieces of information (shapes, properties, features, risks, resources, & processes) becoming manageable (retrievable, in a context, interrelated, reasonable and versionable) supports 3) a flexible workflow and 4) a traceable result (see Figure 4.29).
Sometimes a feasible solution is not easy to generate due to limitations in technology and business logic. For example, representation and exchange of digital product catalogue have been a critical issue in the engineering context (paper C). In an industrial sector, such as cutting tools, vendors, users and system suppliers often use different views and representation methods of information, which makes communication between the parties difficult. The interoperability issue can be solved by system integration with three functional units each of which interprets and presents different types of data (Figure 4.30). The units can be implemented as independent applications or integrated as plug-ins with existing systems.

For system integration, it is possible that no requirement on functions but the interoperability as a quality requirement needs to be implemented in an existing system. The kinematic data exchange for CAD / CAM systems exemplifies this scenario (paper E). Siemens NX and STEP-NC Machine are two systems possessing different exchange structure of a same type of information, the kinematic mechanism. Two plug-ins should be implemented to blend in the existing systems and existing workflows (Figure 4.31). The two plug-ins are named as KIBOS for NX and KIBOS for ST-Machine.
4.6 The conception stage

In this stage, a problem space of the context should be understood at a more detailing level so that designers identify what kind of applications can meet users’ needs before defining how the applications should behave.
Moreover, traceability should be established to justify why requirements are stated (Rubin and Goldberg, 1992). Technically, the previous step has already done a part of the job, because the user goals are often considered as an important part of scenarios (and even use cases). In this stage, the user models are fitted into a more extensive context (Kuutti, 1995), the scenarios. Also, the designers can extrapolate the requirements of functions and content based on the scenarios in a narrative form.

Figure 4.32 A high-level reference model specified in ISO 14649 (author reproduction based on ISO, 2003b).

Usually brainstorming is recommended in this stage because there may be too many inputs and ideas on the carpet. The high-level information models, e.g. the ones in the hierarchical activity models and the reference models, can significantly reduce the chaos. Designers can define the requirements on functions and content based on explicit illustration of activity models at lower levels. The reference models with structured and grouped metadata are essentially a formalized example of IA (Information Architecture) of the application context. Figure 4.32 displays a reference
model at an overview level for information models of computerized numerical controllers (ISO, 2003b). This kind of semantic information is important to form content requirements and functional requirements. The designers may abstract the information models and may incorporate the abstraction with other types of resources to establish a foundation for integration architectures and information architectures.

4.6.1 Scenarios

Software architecture design is recommended to be triggered by scenarios which are brief narrative descriptions for expected use of the new system by the personas with specific goals (i.e. the application context). Normally the scenarios should present the following information (Cooper, et al., 2014):

- In what setting;
- Amount of time;
- Frequency and interruption;
- Teamwork;
- With what other techniques;
- Primary activities leading to goals;
- Expected end results;
- How complex for users with specific skills and use frequency.

Context scenarios briefly imply users expected functions, priority of functions, interface styles and interaction styles. Details about product functions and interactions are not revealed, nor are implementation details. Existing products and technologies should not limit the context scenarios (Cooper, et al., 2014), unless some information is an initial constraint (e.g. for quality assurance activities in some industry sectors, PFMEA is a standard that users shall conform to). For instance, below are context scenarios of risk assessment process based on PFMEA:

1. A temporary team is formed for risk analysis to exploit senior expertise from multiple departments, with a moderator nominated. Latest digital models and previous risk analysis data are prepared, integrated and presented by the system. RPN (Risk
Priority Number) Criteria are set. The interface is usually displayed to the team via some digital hardware (possibly remotely).

2. Quality issues are assessed within the scope of a process plan in a flexible iterative fashion. The assessment may begin with a view of a feature, an operation, an executable step, a setup or a type of resource, although risks will be identified for a process eventually.

3. For each risk being analyzed and specified, the moderator will quickly locate related PPR (Product-Process-Resource) information for the discussion and assign properties and indicators for the risks. The team may need suggestions about naming, description and indicators according to previous risk analysis models.

4. Critical risks in terms of failures are highlighted, with a diagram analysis generated in a real-time fashion. Countermeasures for the risks can be specified in real-time, or in the end of the discussion.

5. Completion of countermeasures or changes in the related information models may trigger revision of the risk analysis. The influences of the changes on PPR models are highlighted, e.g. changes in product design as causes or effects may eliminate, produce, or exacerbate the risks.

6. The results will be exported and archived as an interoperable data set which should be computer-interpretable for control plans and work instructions.

Sometimes, particularly common for system integration, the scenarios are descriptions of enhanced existing workflow. The goal is to blend the developed applications into existing systems to facilitate the interoperability. The context scenarios for kinematic data exchange is one of such cases:
1. In a CAx system, a product design model with geometric and kinematic information is prepared, of which a kinematic chain is completed in an engineering design activity.

2. A function to export/save the design model with both types of systems in a standardized AP242 model is invoked and a name and a location of the exported data set are specified.

3. In another CAx system, the exported AP242 data set is selected and opened/imported to the working area, where kinematic analysis or simulation can be performed on the model which also integrates geometric information.

4.6.2 Requirements

Generally speaking, a requirement is an exhibitable property to practically solve problems (Abran, et al. 2004). In the domain of software engineering, it can be defined as “a need and its associated constraints and conditions.” (ISO/IEC/IEEE, 2011a, more details about application requirements and information requirements in Section 2.2.1) To develop a complex system such as enterprise system integration, requirements are relatively independent of product definition (Garrett, 2010). It is the requirements that define “what” the product should be before jumping to decisions about “how” it should be implemented. System designers have to, in an early stage, understand users’ objectives, business needs, success metrics, state of the art, technical requirements and priorities. This fashion provides flexibility for designers to explore all possibilities beyond technological constraints. Clear requirement definition also addresses validation criteria to evaluate fitness of a design (Cooper, et al., 2014). It avoids an illusion that performance of products can be increased by simply piling up functions and features.

Most requirements can be directly extracted from the scenarios and an ideal format may consist of objects, actions and contexts (Shneiderman and Plaisant, 2004). In this way, the requirements on functions, data and, to an extent, performance can be expressed. Clarity and accuracy are most important for definition of requirements because its high influence of following design activities. Hence, requirement engineers proposed basic
principles to define SMART requirements (Mannion and Keepence, 1995) borrowing from the domain of management psychology (Doran, 1981):

- Specific,
- Measurable,
- Achievable,
- Relevant,
- Traceable.

Based on the context scenario example for risk assessment provided in the previous section, the requirements can be elicited following the same sequence as defined in the scenarios (note that this is a refined version after one prototyping iteration):

- The system records information about the team members, the moderator, time and place during initialization.
- Relevant data for risk assessment is gathered and integrated from related information models or legacy documents, i.e. previous risk assessment results, process plans, product design and production line design, during initialization.
- Criteria of highlighting all indicators of failures (severity, occurrence, detectability and RPN, IEC, 2006) are clarified and recorded, during initialization.
- The system simplifies the above steps by reusing previous settings if the session is triggered by updates.
- Relevant data elements (working steps, operations, risks and features) based on standards on information model of process planning (ISO, 2003b) and PFMEA (IEC, 2006) are integrated and displayed textually and graphically.
- Any relevant data element can be navigated, identified and selected freely from an interface.
- Related PPR information is provided for discussion when a risk is identified.
- Information in terms of modes, effects, causes, detectability and countermeasures are input by the moderator when specifying a risk.
Suggestions regarding naming, description, reasoning and indicators are provided based on previous risk assessment results when specifying a risk.

Whenever a risk item is added or edited, critical risk items are updated and highlighted in a real-time fashion and so is an automatically synthesized report for criticality with diagrams.

The results will be exported and archived in an interpretable way for authors and readers of control plans and work instructions.

Updates of relevant data elements and related models of process planning, product design and manufacturing resources are highlighted and summarized if the session is triggered by the updates.

A revision report will be exported if the session is triggered by the updates.

It can be observed that the application context may result in compromises on whether to specify implementation details. Normally, the technological concern should be addressed as little as possible. However, terms regarding specific technologies may be mentioned in the requirement list, e.g. information models and standardized terminology which have been specified as constraints by certain user groups.

A flow chart with swim lanes (exemplified in Figure 4.33) can be served to display how these requirements are connected, which represents users' needs sequentially. The textual expressions are simplified into phrases in blocks and lines with arrows indicating the sequence of how users accomplish their goals. Although there is a direction for each arrow implying a main path, in most cases of design activities, systems should allow users to go through the required functions back and forth, rather than to follow any specified sequence. The swim lanes provide categorization of the requirements, where the categories indicate a generalized three-stage cycle (analysis, synthesis and evaluation) of any design activities (McNeill, et al., 1998, more details in Section 4.2). The requirements essentially should support all stages in this cycle. Nevertheless, the system integration in this thesis does not always need to support this cycle, when applications are built as an enabling system only for increasing interoperability. This will be exemplified in the following case for kinematic data exchange.
Another example is for kinematic data exchange, where the scenarios are simpler than the risk assessment system. The interoperability as a quality requirement is more concerned, in this case, than any functional requirement. Moreover, no additional function, except basic I/O operations, should be implemented so that the application can easily blend in. This is a typical application with simply functionality but potentially complex data structure. Based on activities models and reference models in the relative information model standards (ISO, 2014a; ISO, 2014b), relevant knowledge about terminology widely accepted in industry can be easily obtained. Here is a list of requirements explored from the context scenarios:

- Completeness and validity of kinematic model is confirmed.
- Special kinematic links (e.g. the workpiece and the cutting tool for a machine tool) are identified.
- The export/save function is chosen by the user.
- The name, the location and the type of the exported/saved data set is specified by the user.
- The data set is exported as configured.
• The import/open function is chosen by the user.
• The targeted data set is located and selected by the user.
• The selected data set is imported.
• The selected model is integrated with other functional units and displayed in a proper module of the existing systems, e.g. in a machine tool library or in a working area.

Again, the following flow chart (Figure 4.34) illustrates how these requirements are related. The three swim lanes show three parts of the development that can be separately implemented as standalone applications or plugins integrated with corresponding existing systems. The three beginning points indicate the three parts are not necessarily executed one by one in a real setting, although together they achieve the interoperability between some designated systems as the implementation goal.

![Kinematic data exchange flow chart](image)

Figure 4.34 Requirements of kinematic data exchange.
The framework stage

The next stage, framework, is where design starts. Scenarios and requirements are translated to a framework which specifies the overall structure of the application software. From the perspective of software engineering, this stage can be seen as a high-level architectural design phase (see more about software architecture in Section 2.4). In this stage, detailed use cases, general description of system components and data models are generated. Use cases in a concrete fashion are greatly useful to generate, refine and validate the framework. It is also important in this stage that developers make best of the knowledge about contexts and information requirements in the activity models and the reference models down to detailed levels, in order to construct a high-level architecture. The following figures (Figure 4.35 and Figure 4.36) are activity models and reference models related with machining operations.

Figure 4.35 An activity model related with machining operations (author reproduction based on ISO 14649 STEP-NC).
This is also a stage where the implementers begin to invest effort on the model mapping process. This mapping process could be troublesome sometimes, because the relationship may not be so clear between data and functions when using information models with less consideration of implementation. Designers should adapt the models to the application scope and create the framework with suitable IA. Obviously, the activity models, the reference models and the use cases have different semantic granularity. Scenarios and requirements established in the previous stage can help to build semantic connection between the models. The provided activity model in the diagram (Figure 4.33) is an example of little explication on its relationship with reference models. Such cases require designers to devote more effort to establish the relationship according to the domain knowledge.

4.7.1 Use cases

Software architecture design requires a structured outline of forms, behaviors, styles and I/O methods, where all the views about functionalities, data structure, workflows of key paths and interface design can be connected. The use case is a methodology that provides such an outline for designers. The use case as a UML (Unified Modeling Language) modeling technique adopted in this study is text-based (mainly), structured and step-by-step descriptions about how a user interacts with a system to achieve a specific goal. Tasks are allocated to users or systems based on the scenarios and the requirements defined previously.
A key path workflow can be presented in a way of concrete use cases (Stone, et al., 2005) to expedite identification of important objects, attributes and actions. The specific goal of concrete use cases can be defined directly from the requirements elicited in the previous section. Work reengineering is performed based on the concrete use cases to clarify concepts in a subject domain. It can be delivered with any decoration method preferred by the designers to indicate relevant information. For instance, one important requirement is to specify a risk by inputting related assessment information (Table 4.1). Objects are highlighted by bold, attributes are highlighted by underlines and behaviors are ignored.

<table>
<thead>
<tr>
<th>User action</th>
<th>System response</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user selects a <strong>failure mode</strong></td>
<td>The system highlights presentation (e.g. the name and the ID) of the <strong>failure mode</strong> in a <strong>failure mode collection</strong>. The system displays the related <strong>process</strong> with parameters, related <strong>features</strong> in the <strong>geometry</strong> of the <strong>product model</strong>, related <strong>resources</strong>.</td>
</tr>
<tr>
<td>The user specifies one or more <strong>risk items</strong> with <strong>failure effects</strong> (for the <strong>final product</strong> or a certain downstream <strong>process</strong>, related to severity), <strong>failure causes</strong> (could be the <strong>failure mode</strong> of lower level steps, related to occurrence), <strong>detections</strong> (related to undetectability) and <strong>descriptions</strong></td>
<td>The system displays real-time suggestions of naming and values of similar <strong>failure effects</strong>, <strong>failure causes</strong> and <strong>detections</strong> in previous <strong>PFMEA documents</strong> and <strong>reliability records</strong> of similar <strong>process steps</strong>.</td>
</tr>
<tr>
<td>The user finishes the specification</td>
<td>The system displays the RPN value and diagrams regarding the input values and RPN in a <strong>real-time report</strong> and puts highlight colors on notable RPN and SOD values according to default or customized <strong>criteria</strong></td>
</tr>
</tbody>
</table>

Table 4.1 Working reengineering on a use case to specify risk assessment.
4.7.2 A design framework

The results of concrete use cases can be conveniently mapped to a design framework (Cooper, et al., 2014). Following the sequence specified in the context scenarios, an abstract presentation of how to organize the data elements is generated in a hierarchical manner. Key elements that may be reused by several scenarios should be highlighted. Generally categorizing the elements contributes to brief understanding of data structure for data modeling in the next stage, in terms of attributes, methods, inheritance and collections.

Two types of elements can be generated from the reengineered use cases: Functional elements and data elements (Cooper, et al., 2014). Each interaction (e.g. a row in Table 4.1) in the concrete use cases forms a function element. The decorated nouns are taken as data elements. The direct mapping result can be preliminary since there may be replications and unrevealed items in the use cases.

The design framework is a foundation to bridge design and construction and to create formal data models for construction. This thesis uses a tree table (can be easily converted to a mind map) to illustrate the design framework with established connections between data elements and function elements of the design framework (see an example Table 4.2). Knowledge gathered in the previous stages and in this stage is synthesized into a hierarchical pattern with at least four levels. Other types of forms are also feasible according to designers’ preference such as indented lists or block diagrams. The four levels are needed to represent the hierarchy of the design framework:

1. Main scenarios
2. Requirements
3. Functional elements
4. Data elements

Note that some elements are repeated in different branches in the table tree because they are being reused in different scenarios. The element called “feature” is an example for the case of risk assessment since it is a
significant part to link process planning and product design. These replications indicate the possibility of integration of functions or scenarios. Still, the objective of the design framework is to identify the elements, so that the items at the first two levels can be simplified and combined to avoid repetition. For instance, in the following Table 4.2, there is a node called risk identification and specification which is a combination of two requirements specified in the previous stage. Another example of the design framework is for kinematic data exchange (Table 4.3), which share the same setting of levels, where the requirements for imports and exports are combined to eliminate some repetitions.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Requirement</th>
<th>Function</th>
<th>Data</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Team gathering</td>
<td>Record team</td>
<td>personnel</td>
<td>Responsibility role</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>moderator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Record time &amp; place</td>
<td>time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>place</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Document preparation</td>
<td>Load core process data</td>
<td>Process models</td>
<td>features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gather and integrate models</td>
<td>Related product models</td>
<td>Process hierarchy features</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>geometry</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>General property</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>classification</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Related product line models</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Name &amp; values</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Criteria in previous risk models</td>
<td></td>
</tr>
<tr>
<td>Criteria setting</td>
<td>Specify criteria</td>
<td>Criteria in previous risk models</td>
<td>Significant values for S, O, D, RPN</td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td>Process item view</td>
<td>Specify criticality presentation methods</td>
<td>Parameters instantiation</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------------------------------------</td>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>View process data</td>
<td>operation</td>
<td>View product data</td>
<td>geometry</td>
<td>classification</td>
</tr>
<tr>
<td>View resource data</td>
<td>resource</td>
<td>Machine tools</td>
<td>line</td>
<td>workpiece</td>
</tr>
<tr>
<td>Integrate views</td>
<td>Feature view</td>
<td>Process view</td>
<td>Operation view</td>
<td>Risk view</td>
</tr>
<tr>
<td>Risk identification &amp; specification</td>
<td>Display a selected process or a selected operation</td>
<td>Display editor of a risk item</td>
<td>Risk item</td>
<td>Failure mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suggest possible values</td>
<td>suggestions</td>
<td>Naming &amp; values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display real-time criticality</td>
<td>Reuse data in synthesis</td>
<td></td>
</tr>
<tr>
<td>synthesis</td>
<td></td>
<td>diagram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Criticality synthesis</td>
<td>Display graphical criticality report</td>
<td>Significant values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------</td>
<td>---------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>countermeasure</td>
<td>Display a selected risk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify countermeasure</td>
<td>Action description</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>personnel deadline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>export</td>
<td>Export report</td>
<td>Standardized model</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information for control plans and work instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>evaluation</td>
<td>trigger</td>
<td>Reuse initialization setting</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Visualize triggers</td>
<td>Changes in documents</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product models</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resource models</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process models</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Countermeasure related</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Update exported</td>
<td>Export updated report</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>risk countermeasures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2 Design framework of risk assessment.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Requirement</th>
<th>Function</th>
<th>Data</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Model verification</td>
<td>Identify geometric models</td>
<td>Id</td>
<td>placement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>parent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identify kinematic models</td>
<td>link</td>
<td>component</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>joint</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Link [2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Placement [2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Limit [2]</td>
</tr>
<tr>
<td></td>
<td>Base</td>
<td>Base link</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Establish link</td>
<td>link</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kinematic chain</td>
<td>Select links</td>
<td>component link</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify links</td>
<td>link</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>General feature</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset I/O</td>
<td>Function selection</td>
<td>Holistically display the function</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify name and location</td>
<td>File name location</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify type</td>
<td>Model type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extension spell</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify whether to display chain info</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution</td>
<td>Export /import as specification</td>
<td>Progress notification</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3 Design framework of kinematic data exchange.

A fifth level may be involved to provide detailed description about attributes or properties of the data elements. For information architects, the data elements and the attributes are important resources. In IA, metadata, i.e. “information about information”, was used to define data elements as the basis of all organizational systems (Wodtke and Govella, 2009). Findability, as a system capacity requirement, is the focus of defining metadata in the web world, but metadata is also relevant for functional requirements. The way to describe and organize metadata is also useful for data elements. Wodtke and Govella (2009) defined three types of metadata: Intrinsic (what the thing is), administrative (the way to handle it) and descriptive (how a user describes it). For findability, descriptive information is most important, but it is not the case for different specific functional elements. For instance, descriptive information is of no use for an element that will not be directly interacted by users. Designers should carefully define data elements and attributes based on the functional needs.
The data elements for some functional elements (the third column of Table 4.2) are omitted since they are already referred to for the same requirements (the second column of Table 4.2). Note that there are still some data elements repeated from time to time for different types of scenarios or requirements and that it looks chaotic for this categorization manner of the data elements according to the functional elements. This thesis recommends a high-level concept model to diagram the data elements and relationships between one another (see an example in Figure 4.37). This example divides the data elements into three engineering domains: Risk assessment, process planning and product design. The division roughly gives holistic views in each engineering domain and it still shows the importance in practice of semantics of information models to support architecting system integration.
4.8 The abstraction stage

The design is detailed in the stage of abstraction, where software architecture for processes, objects and UIs (User Interfaces) are proposed for the implementers. Based on the previous stages (the general scope, the narrative conception and the formalized framework), this stage has been prepared for comprehensive implementation-oriented support. This thesis adopts UML class diagrams to illustrate process models and data models. UI design, often neglected by software engineers, is addressed in this stage as well.

In this stage, information model mapping is also performed between reference models and data schemas, which is the most difficult activity in the mapping. The below Figure 4.38 is a data schema excerpted from the ISO standard for the kinematic topology structure representation. This is a typical building block in the leaf level of a hierarchy to constitute an application protocol (e.g. STEP AP242) that is suitable for meaningful instantiation. It also reuses entities from other building blocks, e.g. a generic schema for representation structures (ISO, 2011) and another generic schema for topological representation (ISO, 2014c). As mentioned before (Section 2.2), this interpretation mechanism eases model development and improves extensibility of the information models. However, to a certain extent, it increases implementation complexity, because understanding this composable structure and performing information model mapping are not software practitioners’ strength, compared with reading manuals, looking up API references and reading sample codes.
A MODEL DRIVEN IMPLEMENTATION PROCESS

This is also a stage to start to prepare a project-specific implementation model (Section 3.4), because relevant interrelated data elements with attributes have been stated in the design framework (Section 4.7.2). The data elements in the design framework suggest what should be encapsulated in an implementation model. The creation of an implementation model should be assisted by corresponding concepts in the data schema, e.g. ISO 10303 STEP p105 ed2 (ISO, 2014a). It also leads to a main part of a data model (Section 4.8.2) in the MVC (Model-View-Controller) pattern.
4.8.1 Process models

It is critical in this stage to keep the software architecture to follow principles of software engineering, IxD and designated implementation platforms. The basic aim to design software architecture is to reduce system complexity with widely applied principles such as high reusability, high cohesion of components, low coupling between components and single points of references. These principles are usually achieved by some generally defined techniques, such as layers, abstraction, encapsulation and modularization. There are patterns of software architectures to guide engineers to solve contextual problems for design systems, such as blackboards, pipes & filters, brokers, reflection and MVC.

As a blueprint for development of interactive systems with heavy content, the passive MVP as a variant of the MVC pattern is recommended for implementation of the information models. This selection is not a mandatory choice, especially when there is no HCI (Human-Computer Interaction) at all for some application contexts. With the passive MVP pattern (Sharan, 2015), the view is absolutely independent of the model, the model controls data and data logic related to the subject domain and the presenter controls logic about interface related to the interaction domain. To follow the convention defined in the classic MVC pattern, the presenter will still be called “Controller” in the rest of this thesis (Figure 4.39).

MVC (Model-View-Controller, an architecture pattern)

MVC has been the most influential architecture pattern in actual use when UI is involved (Dix and Shabir, 2011). There are several understandings or revisions of the MVC pattern since the underlying concept (Figure 4.40) has been

![Diagram of MVP pattern](image-url)
addressed in the late 70s (Reenskaug, 1979). The basic principles used by all MVC variants are the same: 1) Above all, concerns of views (presentation) and of models (representation) are divided; 2) views are difficult to test, so no or fewer logics within view; 3) models have no idea about views, but controllers and views depend on models; 4) models are reusable for different views.

Birth of GUI (Graphical User Interface) required change of this classic definition, because views and controllers were essentially interrelated in most implementations of GUI (Figure 4.41, Fowler, 2006). Decoupling them on purpose made nowhere to implement logics for the view. Hence, inventors of a revision of MVC inserted an application model between the model and the interface to handle these special situations, as Application Model MVC (AM MVC), in 1980s (Figure 4.42). However, in this revision, the controller is still redundant especially since the emerging of Microsoft Windows. MVP as a new version of the pattern redefined the relationship between the roles and became a widely-accepted solution (Figure 4.43, Potel, 1996), with its passive variant (Figure 4.44, Sharan, 2015). Nowadays, many famous patterns related with GUI, web and database adopt the idea of MVC, such as Web MVC (used by JSP Model 2, Struts and Ruby on Rails), MTV (used by Django), Web MVP (used by WebForms), MVVM (used by WPF and Silverlight).
The process design should make sure that all functional elements and data elements defined in the previous stage can be supported by a detailed software architecture. It is most important to pay attention to grouping and assignment of program logic in this step. There are many ways to document architectural design about processes. The UML interaction diagram is recommended by the 4+1 view model, but it can easily take too much space with an increasing number of components and interactions. Therefore, this thesis uses the normal flow chart with swim lanes, integrating the MVC pattern, to describe the processes (Figure 4.45). Although the interaction timing cannot be expressed explicitly, the diagram can still be understandable with clear definitions of components and data flows.
MVC process design – Risk assessment

<table>
<thead>
<tr>
<th>View</th>
<th>Controller</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landing</td>
<td>Initial info</td>
<td>Model collector</td>
</tr>
<tr>
<td>Data viewer</td>
<td>Presentation info</td>
<td>Model manager</td>
</tr>
<tr>
<td>Risk identifier</td>
<td>Invoking request</td>
<td>Risk model</td>
</tr>
<tr>
<td>Risk editor</td>
<td>Identified</td>
<td>Risk modeller</td>
</tr>
<tr>
<td>Data parser</td>
<td>Selected info</td>
<td></td>
</tr>
<tr>
<td>Reporter</td>
<td>Editing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Report request</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filter</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.45 A flow chart to describe the process model.

4.8.2 Data models

To formally apply the data elements defined in the design framework to implementation activities, this stage uses UML class diagrams to convey a programming-oriented description of the data elements. The class diagrams for the data model represent concepts that are a part of the model module in the MVC pattern. In the below Figure 4.46, the designated data elements for the risk assessment example are mapped to a class diagram that will be a part of the model module.

In this stage, most decisions made by designers are related to attributes, data types, naming and relationships. It is possible that some data elements should be neglected temporarily. For instance, the criticality report is considered less related to the model but more related to the view, so it is better to place representations of concepts and logic of the criticality report in the controller module or the view module, rather than the model module.
Obviously, a full view of the software architecture design should not only include the classes for information, but also include those for functionalities that at least in both the controller module and the model module. Getters and setters are most common methods that might be involved in this stage. For instance, in the core class “RiskManager” in the exemplified diagram Figure 4.46. The getters and setters for other classes are omitted in this example.

![Class Diagram](image)

Figure 4.46 A class diagram to model the domain concepts.

### 4.8.3 Wireframe design

Today's IxD community has reached a common understanding that UI (User Interface) design is a systematic engineering project including the design activities previously defined in this chapter: User research, scenario
identification, requirement identification, IA, software architecture, etc. There will also be different levels of abstraction for UI design, e.g. sketch, wireframe, visual design and coding of interfaces (depending on platforms) which can be useful for different contexts. Ideally, most parts of the UI design should be performed by experts in different professions for different types of work products, e.g. interaction designers, software architects, visual designers and industrial designers. However, it is often that applications aiming at interoperability in industry are pressed for time so that such professional support cannot be found efficiently. Therefore, this thesis simplifies the UI design process by integrating the model driven approach, with the wireframe design as the last step of the abstraction stage. The wireframes are generated and indexed by the use cases from the previous stage.

Brown (2010) defined wireframes as “a simplified view of what content will appear on each screen of the final product, usually devoid of color, typographical styles, and images. Also known as schematics, blueprints”. Below are two examples of wireframes (Figure 4.47 and Figure 4.48) for two use cases: Identifying a specific risk item from an operation group and modifying risk items of a specific working step. Information regarding shapes, features and risks are displayed respectively and the relationship between risks, operations and processes are diagrammed as blocks to fit in users’ mental models. The wireframes highlight key functions and behaviors with cursors, which is not possible in reality with many cursors on the same screen (Figure 4.47). The wireframes avoid platform-specific details and aesthetic styles with a mostly black-and-white style which guarantees fast modification and iteration.
Figure 4.47 A wireframe design for identifying a specific risk item.

Figure 4.48 A wireframe design for specifying risk items.
4.9 The construction stage

Although a considerable amount of time has been spent in research, analysis and architecting, the application software cannot come into being without construction. Especially when it comes to system integration, developers should exploit potentials of existing applications and methodologies but bypass barriers caused by technical boundaries.

Based on a complex implementation context, the essential missions here in the construction stage are to minimize construction workloads caused by the interdisciplinary errands and to effectively adapt the developed application software in an existing context. The system integration is usually implemented in a practical engineering context. Most of deliveries are integrated applications rather than standalone ones, in order to reuse existing functionalities and data, or to be integrated into existing business processes. The involvement of multiple types of systems makes the implementation an interdisciplinary process performed by implementers with expertise in various domains.

In this stage, implementation details will be discussed in accordance with functional requirements, which may not be about system integration. Technically addition of functionalities is out of the scope of system integration, so these details are ignored in the previous stages. However, the possibility of demanding additional functionalities is inevitable in practice, due to the extensible modeling architecture. This fact significantly affects the landscape of the construction process for system integration. Therefore, the involvement of other functionalities will be investigated at this stage.

4.9.1 Program design

For this activity, two integration types in the course of model driven system integration should be generalized and discussed, according to implementation experiences of this research: 1) Integrate systems without necessity of implementing additional engineering; 2) the integration projects also demand additional functionalities due to missing links in the channel of interoperation.

The first integration type deals with a situation where existing CAx systems provide the required functionalities, as in the following interaction diagram in UML (Figure 4.49). With this design, users mainly interact with the existing CAx systems and the integrated application is developed only
to enhance the interoperability of the CAx. Typically, existing CAx applica-
tions provide existing interoperable solutions to some extent, e.g. all major
CAD applications are capable to export geometric information in standard-
ized interoperable formats such as STEP AP214, AP203, or AP242. The in-
tegration system is thus a supplement to the existing interoperable func-
tionalities in such CAx applications, to avoid “reinventing wheels”. This in-
tegration design not only enable the developed applications fit into existing
workflows, but also simplifies developers’ workloads by reusing existing
functionalities.

Figure 4.49 An interaction diagram for an implementation integrating with ex-
isting CAx applications.

The other integration type focuses on standalone implementation
more, when there is a lack of both engineering functionalities and interop-
erability in existing systems. Interactive data editing contributes to a large
portion of the functional requirements on the implementation. The integration is essentially developed to add new functionalities, to enhance existing computerized information systems, or to automate a non-digitalized workflow. In such cases, a new workflow should be (and has been in the previous stages) invented and integrated with existing business processes. The information models are applied in the implementation to facilitate more than just interoperability, with broad application context and extensible modeling architecture. However, the system integration and the interoperability are still important because new functionalities need interpretable data input to avoid manual input. Accordingly, the implementers have more responsibility in developing new functionalities which result in obviously a lack of existing interoperable functionality for reusing. With this design, users mainly interact with the developed applications, with input of data sets exported from existing CAx applications (Figure 4.50).

Figure 4.50 An interaction diagram for an implementation as standalone application integrated with users’ existing workflow.
This system design enhances both the functionality and the interoperability and the information models are utilized to facilitate interpretation of the input data and the possibility for other systems to interpret the output data. Hence, the application is still an integrated system as illustrated, but integrated into a workflow rather than specific application software.

Although the manners to design the programs look different in both types, there are still some similarity. In both cases, the developed applications should extract relevant information, interpret models, integrate data and export interoperable data sets. Regarding manipulation of information models, the functionalities of both integration types are similar, apart from requests directly from users or not. Hence, reusing the MVC (Model-View-Controller) concepts adopted for the process modeling stage, behaviors of the implementation can be generalized as the illustrated program design in the following Figure 4.51.

![Diagram](https://via.placeholder.com/150)

**Figure 4.51** A general interaction diagram based on MVC for both types of implementations.
4.9.2 Programming and testing

As mentioned in Section 4.1, this chapter mainly focuses on delivering design decisions at an architectural level. As a result, the discussion about programming and testing, an activity close to implementation details, mainly focuses on synthesizing the deliveries from the previous chapter, i.e. the implementation models (Section 3.4) and the implementation architecture (Section 3.5), with the model component of the MVC pattern. The detailed construction of controllers and interfaces will be described to a limited extent.

The MVC pattern essentially divides focus points to cope with data, presentation and business logic. Different components have different responsibilities and delimited discoverable interfaces so that they are integrate-able. With the MVC pattern, the three subsystems with distinct functions and structure can be decoupled, implemented and maintained independently. This pattern should be fitted into a specific problem-solving context. The context for this scenario is the implementation context (Section 3.3), where the implementers implement applications according to user requirements in an application context.

It is only a well-designed API for data models that can satisfy the users’ requirements. Integrated within the modeling architecture proposed in Section 3.5, the implementation models (Section 3.4) are ready for reuse in an MVC-based software architecture. The implementation model acts as a high-level API encapsulating the generic information models, revealing a core part of the information models and providing supportive functionalities to satisfy the needs of programmers. In the next chapter, an example of the implementation model and two cases using it will be introduced respectively.
Chapter 5
Case studies

While hackers can be very good at designing interfaces for other hackers, they tend to be poor at modeling the thought processes of the other 95% of the population.

- Eric S. Raymond

The case studies are mainly based on the appended papers, where projects were performed collaborating with industrial partners and act as major validations and demonstrations for this study. Moreover, many principles and experiences were inferred or generalized from the projects.

Section 5.1 is mostly based on the paper D for an implementation model. The delivery of the paper D was adopted in paper A, B, C and the licentiate dissertation (Li, 2013b). Section 5.2 describes demonstrations in paper A and E for kinematic data management. Section 5.3 describes demonstrations in paper B and C for cutting tool data management.

5.1 Realization of an implementation model: STEP Toolbox

The introduction of the STEP Toolbox was based on a need to reduce the workloads to implement the ISO 10303 STEP as a standard family for information models. An implementation model was developed to take into account interests of implementers in industry with limited knowledge on information modeling and other technical domains. Besides, there was a research need to rapidly implement new information standards and test the performance in system integration. Hence, simplification of implementation efforts to a level acceptable by practitioners in industry and academia was a major driver of the implementation models.

To develop the implementation model, investigation on the implementation context mattered. Traditionally, implementation of standards followed a unique procedure (Figure 5.1). At first a schema or an AP (Application Protocol) should be selected according to a match of application
contexts. Since an application context of a generic AP was often very broad, implementers always needed to select a relevant and semantically-closed part of the application context based on the description in the AAM (Application Activity Model) to implement, according to the context research of the intended implementation. Then, this selected context would be refined and concretized to information requirements by selecting a suitable part of ARM (Application Reference Model), according to application requirements. Alternatively, a recently-employed modularized modeling architecture required the selection of information requirements not performed on an AP ARM, but on the AMs (Application Modules) constituting the AP, e.g. STEP AP242 (ISO, 2014b). Based on either AP ARM or AM ARM, an AIM (Application Integrated Model) was hopefully generated as a valid data schema for instantiation. Then, implementers should choose a binding way to bind SDAI (Standard Data Access Interface) and AP schemas with a chosen programming language. Schema compilation may be triggered if early binding was chosen. Thus, an API, possibly integrating the data schema was ready for programming. However, a valid data set could not be guaranteed by either a compiled program or an exported data set. Validation was a must for each program to achieve interoperability.

Figure 5.1 A unique process to implement STEP-based standards.

To sum up, this was not a convenient or intuitive process for either professional software engineers or end user programmers in manufactur-
ing industry. An API-based implementation process was often more preferred by practitioners. The API encapsulated the SDAI-based interface and the data schemas. Only concepts related with the subject domain were revealed for implementation, as well as domain oriented functionality support, e.g. coordinate translation. SDAI and model schemas would be hidden, but not be cancelled, because interoperability of the data sets was still a main concern. Besides, the original concepts would be revealed if necessary, for some functionalities beyond the scope of the API. With these design principles, the STEP Toolbox API (Figure 5.2) was proposed in the appended paper D.

Figure 5.2 The modeling architecture using STEP Toolbox API.

As in the illustrated layered architecture, the STEP Toolbox API (the middle layer) acted as a connection between applications (the top layer) and data sets (the bottom layer). There were three sub-layers within API, the bottom sub-layer was the original modeling architecture that was useful for implementation, including the generic data schemas to specify semantics in an application context, the exchange structure to specify syntax and the SDAI to specify access interface. Based on the bottom sub-layer, a middle sub-layer was composed of three parts. The middle part, the model managers, was the central controller of all components in the middle sub-layer and the upper sub-layer and contained general operations for model manipulation sessions, such as initialization for different APs and finalization. The component named as geometry was to encapsulate geometric aspects of the data sets, since the geometric aspects were needed for almost all components in the upper sub-layer. The utilities contained functions to
manipulate information models that might not be provided by the standards, but generally necessary for implementation, e.g. coordinate translation. In the top sub-layer, components to represent different aspects of the information models were provided. These components were usually constituted by a manager and a set of classes directly representing relevant concepts, e.g. kinematic joints and kinematic links.

With the layered architecture, an object-oriented API (see Figure 5.3 illustrating an exemplified documentation of the API) was provided as an intuitive way for the implementers. In the next two sections, utilization of this API in two application contexts will be introduced, integrated within the model driven implementation process (Chapter 4).

![Figure 5.3 Documented STEP Toolbox API by Javadoc.](image-url)
5.2 Kinematic data management

In this case, a project involving kinematic data management is studied with several demonstrations performed by different people with the STEP Toolbox API and the modeling architecture. Moreover, the implementation process introduced in the previous chapter is also integrated partly to enable efficient development.

The first demonstration was presented in the appended paper E. For all the demonstrations, the user study has been used as an example in Section 4.5 and been presented as a proto-persona illustrated in Figure 4.28. The problem could be stated as: For machine tools, there was a lack of a feasible solution to effectively automatically exchange kinematic data integrated with geometric models which was often designed in CAD systems but used in CAM systems; and the vision could be stated as: A system neutral solution was provided with enabling integrated applications for CAD-CAM systems and an exchange structure integrating kinematic data with geometric data. The solution was also detailed in Figure 4.31, where two implementations were introduced as the two enabling systems for interoperability. Then, a three-step context scenario has been provided in Section 4.6 to further elaborate the vision and the requirements for the kinematic data exchange was also presented in Figure 4.34. After the design framework presented in Table 4.3, a high-level process model before using the STEP Toolbox API could be illustrated as Figure 5.4 (see more description in section 5 of the appended paper E). It was definitely not an ideal solution since a lot of technical details about information modeling are involved. Hence, in Figure 5.5 the STEP Toolbox was recommended and included in the solution (see more description in section 6 of the appended paper D).
Figure 5.4 A flow chart for an integrated application to export a data set with full integration of kinematic data and geometric data.

Figure 5.5 A process design enhanced with STEP Toolbox API.
Based on the solution illustrated in Figure 5.5, another demonstration was prepared in a collaboration project with industrial practitioners (section 6.2 of the appended paper D). In this project, the STEP Toolbox API was prepared as developed in Java. However, VBA (Visual Basic for Applications) was a preferred programming language to implement kinematic data I/O for a CATIA system in the partner organization (Figure 5.6). Hence, The STEP Toolbox API was compiled as an JAR (Java ARchive) file invoked by a VBA program and connecting with the VBA program with a temporary buffer file similar as the data buffer shown in Figure 5.5. Hence, the same process design was directly migrated to a new implementation context for a similar application context and the STEP Toolbox API as an implementation model was proven reusable in a different implementation context.

Another project for kinematic error data management (paper A) was also prepared related with the kinematic data management. In this project, new types of representation for kinematic errors were involved, in terms of component errors and location errors. These new concepts demonstrated the high extensibility of the generic information model, i.e. STEP AP242 (ISO, 2014b) integrating the second edition of STEP p105 (ISO, 2014a). Based on this extension, an extended version (section 3 of paper A) was
also designed to revise the implementation model STEP Toolbox API (section 5.3 of paper D). This extended API made it fit into a new application context within a new implementation context which involved interaction with an ASME XML (ASME, 2008) information model.

5.3 Cutting tool data management

Besides kinematic data models mainly used for machine tools, classification mechanism (paper B) and categorization mechanism (paper C) of cutting tool product information were also major concerns in the projects related with this thesis. In particular, based on the classification mechanism, product catalogue exchange was enabled between cutting tool vendors with different catalogue structures and customers with different preferred visualization structure. This situation implied a level of interoperability higher than just semantic interoperability, but to a structural level in practical visualization. Integration with 3D geometric models was also a prioritized requirement which could be satisfied by the extensible modeling architecture of the ISO 10303 STEP standard family.

In the STEP standard family, STEP AP242 (ISO, 2014a), ISO 13584 PLib (Part Library, ISO, 2003a) and ISO 13399 (ISO, 2006a) for cutting tool definition were adopted as major facilitators for implementation. Specifically, the STEP AP242 was taken as an exclusive data schema for instantiation. The high extensibility of AP242 was demonstrated by extensions for cutting tool definition, classification and categorization.

For this demonstration, three integrated applications were proposed to enable an envisioned solution (Figure 4.30) and at least three types of data schemas needed to be prepared, i.e. representation definition of customer view hierarchy, representation definition of catalogue hierarchy and representation definition of catalogue classification. Note that the classification was applied on product classes in paper C rather than on specific products as in paper B. Then, mapping mechanisms were specified between the defined hierarchies, with the support of a standardized cutting tool dictionary defined in PLib (Parts Library, ISO 13584) by ISO 13399 to bridge vendor hierarchies and the customer view hierarchies (Figure 5.7). As a result, three integrated applications implemented the mapping mechanisms (see paper C for more elaborations of the mapping principles and the integration design).
Three prototypes were developed to make use of the defined data schemas and the mapping mechanisms, i.e. an Excel-AP242 catalog creator, a catalogue reader and a customer view creator (Figure 5.8). Examples of vendor catalogues, provided by project partners, in MS Excel spreadsheets were taken directly as a data source. The customer view creator was used to specify preferred visualization structure to view cutting tool product information from multiple vendors. The catalogue reader processed outputs
from the previous two applications and created desired visualization of the catalogue data for the users.

Because of the design of multiple integrated applications, prototype development of this case was at a low detailing level. Most stages of the implementation process specified in Chapter 4 were not exactly followed. The deliveries were used to reveal principles to utilize the extended data schema based on STEP AP242 and other relevant schemas. Still, an implementation model was developed for cutting tool product information representation and classification representation, as depicted in section 3.1 of paper B for infrastructures of this case study. Besides, the implementation model was extended for catalogue data management to facilitate the prototypes in paper C.
Chapter 6
Conclusions

Essentially, all models are wrong, but some are useful.
- George E. P. Box

The model driven approach exploits potential of information models as a basis for system integration and as a key enabler for product realization in an engineering context. This approach needs to be implemented into application software with standardized generic information models in a designated context. Implementation is a bridge to connect information models and application contexts of using the information models.

However, implementation has always been an inherently difficult task (Brooks, 1987). In particular, interdisciplinary efforts are demanded for system integration for CAx systems in manufacturing industry. Implementers may experience inefficiency due to a lack of knowledge and skills. Accordingly, application users suffer from inadequate interoperability. This thesis tackles the problem with two strategies: Enhancing modeling architectures and streamlining implementation process. Both strategies are targeted at efficiency improvement, i.e. to reduce resources expanded to effectively implement the model driven system integration.

For the first strategy (Chapter 3), based on an implementation context, this thesis specifies an implementation model and a modeling architecture to integrate the implementation model. With this solution, the efficiency of using and reusing information models is significantly increased. Namely, implementers can skip time-consuming steps required by the traditional modeling architectures, e.g. model mapping, binding and validation. Meanwhile, the solution keeps the existing way to capture information in a complete, valid and extensible fashion. This solution has been validated by instantly increased implementation efficiency for industrial project partners and in-house research work, which also validates Hypothesis I:
HYPOTHESIS I
Introducing implementation contexts and implementation models in a modeling architecture improves efficiency of system integration.

The second strategy (Chapter 4) is elaborated as an integrated model driven implementation process for system integration. Potential of standardized generic information models is made use of, for highly qualified semantic interoperability and for the similarity in procedure. Stages and activities, focusing on agile architectural decision making, are defined in the integrated implementation process. The process is underpinned by information models as a knowledge base rather than technical constraints. The implementation process has been utilized as an underlying procedural principle for demonstration development in several projects (Chapter 5). Hence, the integrated model driven implementation process with the case studies validates Hypothesis II:

HYPOTHESIS II
Facilitated with existing modeling architectures and HCD (Human-Centered Design) principles, a software implementation process enables effective use of information models for system integration.

This thesis extends the model driven approach to a contextual level in the form of model driven system integration. In Hypothesis I, a contextual layer is added to the traditional modeling architectures, where the model driven approach can be efficiently realized. In Hypothesis II, a contextual process is introduced to fully reuse the traditional modeling architectures, where information models act as a knowledge base to drive the implementation practices.

6.1 Discussion

New understanding of architecting information models
The introduction of two new concepts, implementation contexts and implementation models, provides new understanding of how to design a modeling architecture. These concepts acknowledge fundamental pragmatic difference between data schemas and data sets: The primary users of
data schemas are implementers, and the primary users of data sets are application users. Based on data schemas, implementers develop applications which help application users to access data sets (Figure 6.1). Hence, pragmatically usable data schemas are prerequisite of pragmatically usable data sets. Qualified data sets do not suggest qualified information models. This understanding cannot be clear if model developers stick to application contexts as the basis of modeling architectures. In this regard, model developers should design data schemas that are more usable for implementers to make the data sets more accessible for application users.

![Figure 6.1 Implementers develop applications based on data schemas to make data sets accessible for application users.](image)

**Returning ownership of product data**
A direct result of this study is returning ownership of product data to its true owners, the engineers. Engineering design activities could be performed with different systems by different people. Proprietary lock-in often costs huge frustration to reproduce data manually, or to spend substantial integration cost, where losses and errors are unpredictable. This thesis focuses on increasing performance of system integration for product realization based on standardized generic information models. The information models become a bridge between production engineering and software engineering, rather than a barrier. With efficient implementations, it is possible for industrial practitioners to be easily armed by integrated applications with holistic access to every corner of the data.
Avoiding the tool-driven manner
The model driven approach is a key enabler to help engineers focus on delivering the best design, without the delimitation by any software. For years, engineering applications have been so great tools for engineers that sometimes “it's tempting for people to obsess over tools instead of what they're going to do with those tools.” (Fried and Hansson, 2010) In most of current CAx software, there is an intrinsic template, representing a predefined application context implicitly or explicitly. The template not only draws a boundary for content, but also restricts engineering design procedures. Therefore, any application is likely to limit possibilities of engineering design solutions. In other words, the basic design principle “form follows function” (Sullivan, 1896) is violated. A good tool shall be invisible, that is, shall not limit the ways to accomplish user goals. The model driven approach is a novel solution which helps to integrate and collaborate the strengths of different systems. This study enhances the efficiency to apply the model driven approach, so that missing functionalities can be rapidly implemented and served. In this way, engineers will not be restricted by any individual system or any individual vendor.

Leading demonstrations
Using the results of this thesis, several leading demonstrations were accomplished. They contributed to the understanding of potential of information models and to the development of recently released new information standards, i.e. ISO 10303-242:2014 and ISO 10303-105:2014 ed2. The proposed implementation strategies also contributed to a few research projects, i.e. FBOP (Feature Based Operation Planning), DFBB (Digital factory building blocks) and MPQP (Model-driven Process and Quality Planning). In these projects, industrial partners were benefited of the work of this thesis for efficient implementation of system integration.

Being applicable besides system integration
The results of this study are also applicable for other scenarios that need information exchange, e.g. SOA (Service-Oriented Architecture) and cloud computing. A modeling architecture is for sure an infrastructure to support an SOA. Interoperability is the first quality attribute that should be taken
care of for SOA (O’Brien, et al., 2007) and it is the interchangeable information leading to interoperable services (Brownsworth, et al., 2004). The information models, representing product information, have been proven, in this thesis, as an enabling leverage to interface general engineering applications and is capable to be extended for use in SOA. The system neutral information models guarantee a formalized contract that enables services discoverable, independent and reusable. Moreover, high agility of services can be assured by the implementation strategies proposed in this thesis, in that high portability of a formal contract is provided. Hence, the fundamental business goals of SOA can be assured, i.e. reduced TTM, integrated business processes and reusing legacy systems as services (O’Brien, et al., 2007). Cloud computing (Mell and Grance, 2011) can be seen as a specialized SOA with emphasis on networking and resource pooling and cloud manufacturing is a manufacturing version of cloud computing (Xu, 2012). Pallis (2010) identified three types XaaS (Everything as a Service) layers for different user groups. Although a top layer was put on a SaaS (Software as a Service) layer as “User front end”, there are small but critical user groups relying on PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). Compared with plain SOA, PaaS and IaaS are additional topics concerning development of cloud computing and cloud manufacturing. In particular, implementers are direct users of PaaS where effective system integration is required to implement SaaS. For information models as an infrastructure resource of PaaS, portability and extensibility are also crucial capabilities for efficient implementation.

6.2 Limitations

*Necessarily high technological readiness*

Development, deployment and utilization of computerized information systems are greatly affected by technological readiness of organizations and individuals. Although the model driven approach may reduce the workloads of immediate stakeholders to some extent, there is a progressive procedure to take effect. It is the job of engineering users and organizations to accept and embrace the technology. The content-rich systems (Section 2.5) require not only individual technological readiness (Parasuraman and Colby, 2015), but also an organizational level of technical infrastructures
in the supply chain. There should be adequate applications or services to provide functionalities to access useful digital information.

Object Oriented (OO) paradigm

In Chapter 4, this thesis adopts the Object Oriented (OO) paradigm to architect and construct the system. However, the presented methodology, which employs information models as a base for system integration, could be applied to any implementation paradigm. The two hypotheses of these theses are targeted on practices in manufacturing industry and in IT industry respectively. The deliveries of engineering activity in both industries is shared: Documentation, after which the effort is handed to manufacturing team (Reeves, 2005). The information models play a unique role in both practices: As a base for innovation (Nielsen, 2003) to map from one reality (requirements) to another reality (designed artifacts). This role as in general engineering activities does not change whether the implementation paradigm is OO or not.

File-based integration scenarios

Architecting system integration is to explore an appropriate way to achieve interoperability between heterogeneous systems which can be technical units or business units. There are many styles to achieve system integration at an architectural level: File transferring, shared database, sockets, remote procedure call, messaging, etc. (Hohpe and Woolf, 2003) The style mostly discussed in this thesis is file transferring style which is the most common data sharing method (Roshen, 2009). However, the solution should not be limited to this one, because developers are faced with the same challenge no matter which style is selected. It is the challenge this thesis addresses: System integration in an unpredictably changeable environment. The issues are composed of 1) technical aspects: Unpredictable system performance, unpredictable system availability and evolving technologies and 2) business aspects: Changing business needs, changing regulations and changing intra- and inter-organizational structures. This thesis utilizes different techniques (e.g. standardization, information models, HCD, IA and MVC) based on the model driven approach to manage and minimize impacts from the constantly changing factors.
Limited context research

In-depth study of contexts, in terms of behavior patterns of users and systems for engineering design activities, is a missing area in both the domains of software engineering and HCD. This study identifies this issue and uses the implementation context to describe the problem space. A context is certainly useful, but it has an enormous scope. Technically a context should be a formal complete definition of a specific space-time where stakeholders are connected with a (probably envisioned) product. This definition is impossible to be derived completely and validly. On the other hand, there is a large amount of information in the definition that is irrelevant. Discerning what is relevant is the reason why designers have to research contexts, model users and analyze requirements. Although several industrial projects were involved in this study, there have been few opportunities to observe and learn from real application users and implementers.

6.3 Future work

The above limitations are certainly good starters for future work. However, there are potentially more important results that can be generated based on the following discussion. Especially, pragmatic concerns about high-level business requirements should be addressed from different perspectives. Knowledge from standards in other areas is useful for further research. Moreover, the standardized information models are widely used in building industry for interoperability as well. The fundamental idea is extremely similar to ISO 10303 STEP and the EXPRESS syntax is reused directly. However, remarkable work has been done for implementation-ready standardization and confusing terms identified in this thesis such as application contexts and information requirements are dropped. A cross-industry research could be beneficial for these two similar domains.

Integration aspects besides technology

The introduced solutions are potentially promising to overcome practical obstacles to adopt information standards from other perspectives more than technology. The direct driver of this thesis is a performance problem depicted technically, i.e. efficiency to implement system integration. However, “tools by themselves do not promote success; the proper use of the
tools does” (Putnik and Putnik, 2010). Business processes, culture and values are critical non-technological factors in relation with interoperability (Grilo and Jardim-Goncalves, 2010). Many researchers have investigated these factors, whose ideas could be borrowed here. For instance, Kiviniemi, et al. (2008) attributed delay of deploying BIM (Building Information Model) in AEC (Architecture, Engineering and Construction) industry to a triangle dilemma (Figure 6.2). Market demands, software supports and measured benefits together made a paradoxical loop that was hard to break. This is applicable for product realization in manufacturing industry as well. Tassey (1999) shared a similar viewpoint, based on general influences on business and concluded three causes of implementation failure: 1) Non-appropriability of benefits, 2) technical and market risks and 3) a lack of unbiased expertise. Gielingh (2008) also observed poor industrial uptake of STEP-related standards and imputed it to three causes: Low business motivation, low pragmatic readiness and legal aspects. It will be a new direction for developers and researchers on information standards to study how to take advantage of the result of this thesis for these high-level obstacles.

Figure 6.2 “Paradoxical loop” of implementing information models (author reproduction based on Kiviniemi, et al., 2008).

Future development of information standards
This study is also an incentive to raise concern of implementation readiness in future development of information standards. Usefulness of the
standards greatly depends on its usability in implementations and utilization of applications and all applications start with implementations. There are several possibilities to increase implementation readiness formally:

- Develop standards with the focus on effective implementable solutions instead of effective standardization of concepts.
- Reuse and standardize the idea of conformance classes as functional subsets of schemas for specific implementation contexts.
- Keep a simplified implementer-oriented view in the modeling architectures, e.g. keep a small number of layers revealed; use the reference models as additional support rather than constraints.

Many standards have made efforts toward implementation readiness. In some APs (Application Protocols) of ISO 10303 STEP, conformance classes are prepared as subsets of standardized schemas suitable for implementation of specific scenarios. In particular, PLCSSlib (OASIS, 2016) was specified to increase user experiences for implementers, based on a standard for Product Life Cycle Support (ISO 10303-239 PLCS, ISO, 2005). The IFC (Industry Foundation Classes, ISO, 2013) standard is also a decent comparable example of switched concern from concept standardization to a readily-implementable solution (Laakso and Kiviniemi, 2012). Moreover, an enormous amount of effort has been put into developing implementable IFC, e.g. an implementation guide directly for implementers (Liebich, 2009), IDM (Information Delivery Manual, ISO, 2016b) and MVD (Model View Definition) defined according to IDM. An MVD acts as additional (not mandatory) reference models but only specifies a useful subset of IFC for instantiation. This flat architecture facilitates comprehensible initial implementation with “useful minimum” scopes (Hietanen and Lehtinen, 2006). It does not require implementers translating business requirements from contexts to models via sophisticated mapping. The idea of MVD can be a helpful reference to implementation models or any information models that will be implementation oriented.

**Implementation guidance**

Domain-specific programmer-oriented implementation guidance is also needed for standardized information models. Formal data schemas are like codes in an API, for which the most important characteristic should be
“read like a book” (Gillis, 2016). Unfortunately, just like codes, data schemas are also written with unusual syntax and domain-specific semantics, but still should be readable by implementers. This fact demands model developers’ effort on proper implementation guidance:

- Fit with implementers’ existing workflows and mental models.
- Provide interpretation guidance on domain-specific semantics.
- Provide tailoring strategies based on implementation needs.

Implementers may be already familiar with implementing standards for system integration or interoperability, which implies the existing workflows and mental models that can guide implementation. Nonetheless, STEP and related standards are different from common data standards to achieve interoperability at a machine level or a syntactic level, e.g. HTML, SMTP, and TCP/IP. Standardized information models usually are targeted at a semantic level, which implies different implementation conventions. Semantic heterogeneity was a cause for the most difficult problems of data integration, which was considered unsolvable for decades (Ziegler and Dittrich, 2004). However, there may be no generic solution, but domain-specific solutions have been devised in different standards, such as STEP and IFC. Implementation of the domain-specific solutions can be a strange experience for common implementers. For instance, partially implementing a standard, i.e. “embrace but omit”, to achieve syntactic interoperability may cause issues on standard integrity (Egyedi, 2007). On the other hand, information models standardized at a semantic level are meant to be tailored. An aggregate model for a holistic view and discipline models for specific-domain views are both useful for product data management (Figure 6.3, Van Berlo, et al., 2012). Hence, specialized guidance may be placed for implementers who are used to implement full web-based standards.
CONCLUSIONS

Figure 6.3 Discipline models and aggregate models in different stage of a product lifecycle (Van Berlo, et al., 2012).

HCD in information standards

This thesis might be an inspiration of human-centered information standards. “Technology is worthless - even dangerous - if we don't pay attention to the human aspects of both its use and its construction.” Gause and Weinberg (1990) pointed out a common pitfall faced by engineers in our times. BOTH use and construction of engineering applications are executed by human beings and the focused concern is always from human beings, implementers, users, designers, customers, etc. Standards that are used by implementers are also one type of these engineering applications which should be optimized with human centered design:

- Build decent information architectures to increase findability and ease of navigation.
- Design modeling architectures toward both end-user programmers and professional programmers.

Humankind and machinery excel in different aspects. The invention of machinery is driven by a desire to substitute intolerable operations of human intellect and to relieve “fatiguing monotony” (Babbage, 1822). When machinery is designed to interact with human beings, it should maximize performance of both parties, based on strengths and weaknesses of
the human information processing system (ISO, 1999). For most consumer products, traditional usability, in terms of effectiveness, efficiency and satisfaction, has become a “dissatisfier” (Jordan, 2000) that users take for granted. Nevertheless, the domain of information modeling has a different landscape: There is a huge gap regarding human-centered enhancement of information models. Standardized generic information models are not usually designed for usability, as an enabling system shall be intensively comprehended and used by implementers.
Definitions of relevant terms

Basic terms

**Application software**: "Software that is specific to the solution of an application problem" (ISO/IEC, 2015).

**Architecture**: Underlying structure of a system (Brand, S., 1995). "Fundamental concepts or properties of a system in its environment embodied in its elements, relationship, and in the principles of its design and evolution" (ISO/IEC/IEEE, 2011b).

**Computerized information system**: A "human-machine system that provides information to support the operational, managerial, analytic, and decision-making functions" based on the use of computers (Krogstie, 2012; Falkenberg, et al., 1998).

**Context of use**: "Users, tasks, equipment (hardware, software and materials) and the physical and social environments in which a product is used" (ISO, 2010b).

**Data**: A representation of information in a formal manner suitable for communication, interpretation, or processing by human beings or computers (ISO, 1994).

**Information**: "Facts, concepts, or instructions" (ISO, 1994). "Knowledge concerning objects, such as facts, events, things, processes, or ideas, including concepts, that within a certain context has a particular meaning" (ISO/IEC 2382:2015).

**Information architecture**: "<Human-centred> structure of an information space and the semantics for accessing required task objects, system objects and other information" (ISO/IEC, 2010, Section 2.5).

**Integration architecture**: Structure, relations and rationales of system components for system integration, decided at an architectural level.

**Interactive system**: "Combination of hardware, software and/or services that receives input from, and communicates output to, users" (ISO, 2010b).

**Manufacturing**: "The entirety of interrelated economic, technological and organizational measures directly connected with the processing/machining of materials, i.e. all functions and activities directly contributing to the making of goods" (CIRP, 2004).

**Model**: A collection of representations to describe an existing or a future system (Blanchard and Fabrycky, 1990).
**Requirement**: "Statement which translates or expresses a need and its associated constraints and conditions" (ISO/IEC/IEEE, 2011a).

**System integration**: A process to progressively assemble "system components into the whole system" (ISO/IEC/IEEE, 2010).

**Tool-driven manner**: A problematic way to complete engineering goals greatly depending on tools in use with limited functions and content.

**Terms about information models**

**Activity model**: A model to represent an application context, in terms of processes, information flows, and functional requirements, which is usually to state scope of a data schema.

**Application**: "A group of one of more processes creating or using product data" (ISO, 1994).

**Application context**: An environment where product data is used in a specific application (a modified version based on ISO, 1994).

**Application protocol (AP)**: A specification of a data schema satisfying an application context (a modified version based on ISO, 1994).

**Data schema**: An information model to specify representation of information in a particular context.

**Data set**: An information model instantiating a data schema or a conforming segment of a data schema.

**Implementation context**: An environment where implementers use an information model to implement an application.

**Implementation model**: An information model that encapsulates one or more data schemas, with additional supportive functions for data set manipulation, to satisfy information requirements in an implementation context.

**Information model**: "A formal model of a bounded set of facts, concepts or instructions to meet a specific requirement." (ISO, 1994)

**Information requirement**: Information that must be represented in a data schema.

**Model driven**: Keeping information in context, versionable, associable and retrievable in information models (Nyqvist, 2008); using information models to integrate product data from different sources, to create a productive environment and to enhance business competitiveness.
**Modeling architecture**: An architecture to specify development and use of an information model.

**Model driven system integration**: An integration practice to facilitate interoperability between disparate systems based on the model driven approach.

**Product data**: "A representation of information about a product in a formal manner suitable for communication, interpretation, or processing by human beings or by computers" (ISO, 1994), including data for lifecycle management, manufacturing, design, etc.

**Reference model**: An information model to represent information requirements of a specific application context (ISO, 1994).

**Qualities pursued in this study**

**Effectiveness**: "Accuracy and completeness with which users achieve specified goals" (ISO, 2010b).

**Efficiency**: "Resources expended in relation to the accuracy and completeness with which users achieve goals" (ISO, 2010b).

**Extensibility**: An ability to "continue to take advantage of new and innovative technique" (Al-Timimi and Mackrell, 1996). "The ease with which a system or component can be modified to increase its storage or functional capacity" (ISO/IEC/IEEE, 2010).

**Interoperability**: "The ability of two or more systems or components to exchange information and to use the information that has been exchanged" (ISO/IEC/IEEE, 2010).

**Portability**: An ability to "move data among applications" (Al-Timimi and Mackrell, 1996). "The ease with which a system or component can be transferred from one hardware or software environment to another" (ISO/IEC/IEEE, 2010).

**Pragmatic interoperability**: The ease to interpret meanings of exchanged messages by human audience.

**Semantic interoperability**: The ease to interpret meanings of exchanged messages by systems.

**Syntactic interoperability**: The ease to interpret formats of exchanged messages by systems.
Usability: "Extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (ISO, 1998b).
Bibliography


Appended papers