A NEW FORMAL AND ANALYTICAL PROCESS TO PRODUCT MODELING (PPM) METHOD AND ITS APPLICATION TO THE PRECAST CONCRETE INDUSTRY

A Dissertation
Presented to
The Academic Faculty

By

Ghang Lee

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in the
College of Architecture

Georgia Institute of Technology
December, 2004

A NEW FORMAL AND ANALYTICAL PROCESS TO PRODUCT MODELING (PPM) METHOD AND ITS APPLICATION TO THE PRECAST CONCRETE INDUSTRY

Approved by:

Charles M. Eastman, Advisor
College of Architecture, Georgia Tech

Godfried Augenbroe
College of Architecture, Georgia Tech

Dr. Shamkant B. Navathe
College of Computing, Georgia Tech

Dr. Martin Hardwick
Computer Science, RPI

Dr. Rafael Sacks
Civil Engineering, Technion

June 14, 2004

Product modeling is science, not art.

Chuck Eastman

ACKNOWLEDGEMENTS

This dissertation would not have been written without the support and feedback of many people. First and foremost, I would like to thank Chuck Eastman, my advisor and mentor. He is a warm-hearted advisor as well as a passionate and creative scholar. Without his feedback and support, I would not be able to begin and finish my Ph.D. study. This thesis was initiated by his question during a class. While developing GTPPM described in this dissertation, there were many intellectual challenges. His insight, experience, and advice helped me a lot in finding some possible answers. I also would like to thank his wife, Mary Claire Eastman for her kindness and caring for my family.

Thanks to my thesis committee members for helping me structure and further develop this thesis. I am much honored to have very renowned scholars in product modeling as my thesis committee members. Prof. Fried Augenbroe in Building Technology was my co-advisor. His knowledge in process and information modeling and experience broadened my view towards process and product modeling. Dr. Sham Navathe in Computer Science at Georgia Tech was my minor advisor. His lectures on the fundamental database theories helped me clarify many data modeling issues. Dr. Rafael Sacks, an external reader, is a senior lecturer of Civil and Environmental Engineering at Technion Israel Institute of Technology. He's been deeply involved in the development of GTPPM since 2001. Without his feedback and encouragement, I would not be able to work constantly on this research topic for several years. Dr. Martin Hardwick is a professor in Computer Science at the Rensselaer Polytechnic Institute and the CEO of

STEP Tools. I thank him for his interest and detailed feedback and also for sparing his busy time to attend my thesis defense from Troy, NY.

There are several people that I owe acknowledgements for giving me thoughtful and sincere comments on my work. Thanks to Robert Amor and Anders Ekholm.

In parallel to my thesis, I worked on a project to develop an intelligent 3D parametric CAD system for the North American precast concrete industry. The project was called the PCSC (Precast Concrete Software Consortium) project. Through the PCSC project, I was able to test and evaluate my new "process to product modeling (PPM)" method, which is my thesis topic. I am indebted to the PCSC (Precast Concrete Software Consortium) members and Tekla for their support, help, and collaboration. I was very impressed by their passion for the project. I am especially thankful to Hans Klohn, Mark Aho, David Orndorff, Lee Tanase, Dave Mahaffy, Mike Hutchinson, Skip Wolodkewitsch, Karen Laptas, David Campbell, Dave Bosch, and David Fiedler from the PCSC, Jason Lien and Harry Gleich, the former members of the PCSC, and Ragnar Wessman and Pertti Alho at Tekla. Bill Heeps, Information System Manager at High Concrete was not a member of the PCSC. But without his help, I would not be able to evaluate my method.

In 1995, while I was working at Kumho, I was fairly ignorant about *Design Computing* as a field of study. I am grateful to Dr. Hyuk Song and Hanmin Lee, my former bosses at Kumho, for introducing Design Computing to me and giving me a chance to build my career in this area. I also thank Dr. Sungsuk Go, Dr. Jongsung Im, Rakgi Choi, Jungkyu Lee and other colleagues at Kumho for their support.

I thank Dr. Ji-Hyun Lee at NYUST, Taiwan (previously at Carnegie Mellon) for directing me to Chuck Eastman at Georgia Tech when I was looking for the best place to pursuit my Ph.D. study.

I am also thankful to my former advisors at Korea University, Dr. Jungduk Lee, Dr. Namchul Ju, and Dr. Kyungin Kang for their advice and consistent support even ten years after I left school.

I also like to thank the Ph.D. Program at Georgia Tech. First of all, I'd like to thank Tom Galloway, Dean of College of Architecture, and Chuck Eastman (again), Director of the Ph.D. Program. Thanks to their support, I was able to attend many conferences while I was staying at Georgia Tech and was extremely lucky, as a Ph.D. student, to become a keynote speaker and a session chair at a couple of conferences.

However, without good friends, I would not have been able to enjoy my life at Georgia Tech. I thank Frank Wang (and Liyen), Fehmi Dogan (and Jenny), David Craig, Seokjoon You (and Saetbyul), Donghoon Yang (and Eunyoung), Jaemin Lee (and Nan-a), Saleem Mokbel, Weiling He, Elif Sezen Yagmur, Yan Zhang, Hyeonjoon Moon (and Eunkang), Cheolsoo Park (and Sujin), Pegah Zamani, Mahbub Rashid, and others for their feedback and friendship.

I usually don't like Hollywood stars listing names of their friends and sponsors at an Academy Award. But I cannot just skip people who keeps reminding me that the earth is a wonderful place to live: Kyuman Park, Yu-kyung Bae, Giyoung Kim, Jung-a Kim, Byungho Kang, Jung-a Lee, Hannui Lee and Barbara, Youngjoon Simon Lee, Donggi Namgung, Seung-Jong Park and Youngjin Chon, Hyeonjoo Park, Jooyun Chung, Seungyun Gong and Yujin Lee, Jaesung Ryu and his family, Yum Park and Younkyung

Lee, Seong-Hyeak Won and Seungwon Shin, Jeehoon Park, Hyunghoon Kim, Hun-Hee Cho, Youngchel Yum, Sanghyuk Son, Sejin Chon, Sujin Jang, Hokyu Hahn, Namho Park, Hyungchang Lim, Shinho Kim, Kanghee Lee, Heesuck Henry Lee, Woong Lee, Youngjoon Oh, Youngjoon Park, Kyoungwook Seo, Marim Kim, Youngjae Kim, Jinsoo Lee, and many others.

I also thank my middle school teacher and mentor, Soonjae Lee for her over twenty years of support.

Many thanks to my parents, Professor Kiyong Lee and Director Jungja Ha, for always trusting my decisions and for raising me as a positive and optimistic person. Thanks to my sisters, Sue-en and Juen, for always being my friends and supporters. Thanks to my parents-in-law, Mr. Mooyoung Park and Mrs. Chunja Kwak, for raising their daughter as a beautiful and wise lady.

I cannot thank enough my wife, Sungjin Park for her patience and support, and Gio and Gia for reminding me how beautiful and delightful the world is. Sungjin, Gio, and Gia have always been the motivation of my life and they will be.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A product (data) model is a formally structured schema of some subset of the information that is generated, modified and deleted throughout a product's lifecycle. Product models are being developed in many manufacturing, construction and industrial domains to facilitate automation of activities, electronic communication and re-engineering of engineering processes. The current standard product (data) modeling process relies on the experience and subjectivity of data modelers who use their experience to eliminate redundancies and identify omissions. In order to ensure correctness, their decisions are validated via a time-consuming process of national and international voting, e-mail and face to face meetings. As a result, product modeling becomes a social activity that involves iterative review processes of committees.

This study aims to develop a new, formal method for deriving product models from data collected in process models of companies within an industry sector. The theoretical goals of this study are to provide a scientific foundation to bridge the requirements collection phase and the logical modeling phase of product modeling and to formalize the derivation and normalization of a product model from the processes it supports. The long term practical goal is to greatly reduce the time and cost of producing a product model from the current 5 to 10 years to 1.5 years or less. Another practical benefit will be to allow companies to better plan and integrate their operations using the resulting product model. To achieve these goals, a new and formal method, *Georgia Tech Process to Product Modeling (GTPPM)*, has been proposed. The basic approach is to

bind *process* and *product data modeling* together and to develop a product data model that is sensitive to its various applications (processes).

This method eventually intends to support the ISO STEP effort. ISO STEP (STandard for Exchanging Product data) is an international effort to develop standard product models. The equivalent concepts to process and product models of ISO STEP are respectively Application Activity Models (AAMs) and Application Reference/Requirements Models (ARMs). Currently EXPRESS is the standard ISO STEP data modeling language and IDEF0 is the standard AAM language. However, an AAM and an ARM are linked implicitly and abstractly. In order to provide a mechanism to tightly bind them together, several research questions should be answered. The research questions are:

1) What is the process semantics that is required to elicit processes and information necessary and sufficient to derive a product model?
2) How to specify required information in a machine-readable format
3) How to resolve the naming issues (a.k.a. the '*nym*' issues: e.g., synonyms and homonyms) and the conflicts between company-specific vernacular terms and a consistent machine-readable terms
4) How to validate the consistency of information captured in a process
5) How to derive a product model from the collected process information
6) How to validate the well-formedness of the derived product model and normalize the derived product model

7) How to integrate (or harmonize) product models into one unified model when several different product models are derived from different processes about the same product.

GTPPM consists of two modules. The first module is called the *Requirements Collection & Modeling (RCM)* module. It provides semantics and a mechanism to define a process model, information items used by each activity, and information flow between activities. Thirteen process-modeling components have been defined for capturing process semantics and information flow. In order to specify information items used by each activity, a mechanism, called an *information menu*, has been developed. It structures and restricts a way to specify *information constructs (ICs)* based on rules defined using a *context-free grammar (CFG)*. Information constructs (ICs) are formally defined information items and represents domain semantics. The logic to dynamically check the consistency of information flow within a process also has been developed.

The second module is called the *Logical Product Modeling (LPM)* module. It integrates, decomposes, and normalizes information constructs collected from a process model into a preliminary product model. Nine *design patterns* are defined to resolve conflicts between information constructs (ICs) and to normalize the resultant model.

These two modules have been implemented as a Microsoft Visio® add-on. The tool has been registered and is also called GTPPM®. The method has been tested and evaluated in the precast concrete sector of the construction industry through several GTPPM modeling efforts. The GTPPM was first deployed by fourteen precast producers in the North America in analyzing the sales, design, engineering, production, and shipping processes and information flow in the precast concrete industry. Based on the

analysis results of the first attempt, three more test case models were developed. Three product models and one integrated product model were automatically derived from the three GTPPM models. One product model of a company was compared with the existing Enterprise Resource Planning (ERP) system of the same company. The integrated model was evaluated using the precast concrete extension of an existing standard product model (i.e., PCC-IFC) as a benchmark.

A product model generated by the current GTPPM method is by no means complete. An automatically generated product model will not include roles, data type, cardinality, and the WHERE, DERIVE, and RULE clauses. Those should be added and modified manually. The logic for automating those processes can be developed further in the near future.

By using GTPPM, a complete set of information items required for product modeling for a medium or a large industry can be collected without generalizing each company's unique process into one unified high-level model. However, the use of GTPPM is not limited to product modeling. It can be deployed in several other areas including:

- workflow management system or MIS (Management Information System) development: Information required for processing an activity, passed to succeeding activities, and fed back to previous activities can be defined.
- software specification development: A detailed definition of engineering functions and processes can be developed, which will allow further development of software in the engineering and design areas.

- business process re-engineering: A process model with specific information items can be used for reengineering of an organization like other process models.

Also any form of a data model defined in EXPRESS can be read into GTPPM as an *information menu*. Using this function, GTPPM can be used to update or validate an existing product model by reading in an existing product model as an information menu. It can be also used to develop *conformance classes* (i.e., valid subset models) of an existing model.

We hope that this work will impact American and international standardization activities (e.g., ISO efforts) to develop product models. By developing new formalisms for product modeling, the proposed method is intended to build a formal and scientific foundation for work in a field that is currently a craft, allowing systematic improvement.

CHAPTER 1

INTRODUCTION

1.1 WHAT IS A PRODUCT MODEL

The information involved in design, engineering and manufacturing of each product class involves many specialized entities, various types of aggregation, attributes with specialized meaning and functional relations. A *product (data) model*[1] is a formally structured schema of such product information that is generated, modified, and deleted throughout a product's lifecycle. Defined as an integration and exchange standard, it is an electronic medium to share and exchange product information among heterogeneous systems within an organization, or more widely within/across industries. A product model has distinctive characteristics from other data models:

1) It includes *complex geometric information*, defining the shape of each component of the product, and also the shapes of different levels of component composition.
2) The geometry is partially derived by *the product's intended functions*. These functions of the product are represented along with the topologies that enable them, as well as the behavioral analysis results used to determine properties of the product, partially capturing the product's intent and rationale.
3) A product is *manufactured or constructed*. The information required to fabricate, assemble, test, and manage the product are also included.

[1] In this thesis, the terms a 'product model' and a 'product data model' are used interchangeably. Also a 'data model' and a '(data) schema' are used interchangeably.

To date, over 30 product data models have been developed within the International Standards Organization - Standard for Product Data Exchange (ISO-STEP 10303) standards (ISO TC 184/SC 4 1994) and there are a growing number of industry-based product models developed outside of the ISO organization, but using the same technology, tools and procedures (CIMSteel Integration Standards Release 2 2002; IAI).

Product model schemas are large and multifaceted, reflecting multiple complex semantic domains. For example, the CIMsteel product model used in the structural steel industry (Crowley & Watson, 1999) has 731 entity types and a scope covering the design, analysis, shop detailing and fabrication of steel structures for buildings. Currently, it is supported by twelve applications. Other example domains for which product models have been developed include NC tooling (ISO TC 184/SC 4 1996), sheet metal design processes (ISO TC 184/SC 4 1999; Jurrens 1991), piping (ISO TC 184/SC 4; Palmer and Reed 1990), process plant spatial layout (ISO TC 184/SC 4 2001), electronic assembly and packaging design (ISO TC 184/SC 4 2001) etc. While significant effort has already been applied to the development of product models, many engineering and production domains are still evolving their IT infrastructure and have not yet developed corresponding product models. Also, product models are live, not static, and require updating as new technologies and concepts are integrated into a manufacturing or design domain. Thus the benefits of improving the methods used in product modeling would have significant impact.

1.2 A STANDARD METHOD FOR PRODUCT MODELING AND ITS DRAWBACKS

The current method employed in all current product modeling efforts is based on the ISO-10303 STEP languages and methods (NIST 1993, 1993). The STEP name for a product model of each domain is an Application Protocol (AP). The STEP includes

standard procedures that correspond to the ANSI/SPARC three-level database architecture: i.e., a view, a logical model, and a physical model. The procedures begin by defining the scope and processes to be supported, by defining a process model of the domain of discourse, (called an Applications Activity Model (AAM)). STEP uses IDEF0 (Integration Definition of Function Modeling) to define the AAM. It shows "the engineering process context in which an AP will be used (VTT Building and Transport 2002)". From the AAM, a view of the information domain (called an Application Requirements Model (ARM)) is defined using one of a set of conceptual modeling tools. (ISO STEP currently endorses NIAM (Nijssen and Halpin 1989), IDEF1x (NIST 1993) and EXPRESS-G (ISO TC 184/SC 4 1994).) An Application Requirements Model is then refined and elaborated into an Application Interpreted Model (AIM, which is a logical model of the information domain). EXPRESS is the product modeling language universally used in such efforts (ISO TC 184/SC 4 1994; Schenk and Wilson 1994; VTT Building and Transport 2002). The initial AIM is then refined to integrate standard data model resources for representing standard, cross-discipline concepts, such as geometry, units and measurements, organizations, and so forth. The product model must support a variety of uses, centered around queries, access, and management. These often require data about the data, or metadata, needed for data management uses. Later, AIM can be implemented as a physical model through the Standard Data Access Interface (SDAI) (ISO 10303 Part23, 2000). Table 1.1 on the next page maps the STEP models and the ANSI/SPAC three-level data structure based on Andrew Crowley's five-level structure on p. 40 of (Crowley 1998) and other references (Eastman 1999; Elmasri and Navathe 2000; ISO TC 184/SC 4 1994; NIST 2002).

Product models are currently developed as a joint undertaking of domain experts and product model experts, relying on committee reviews and convergence. The domain experts rely on natural language to describe their requirements. The product model experts first use process modeling languages and tools to define the scope of the domain (the AAM) and then conceptual modeling tools to define the concepts in the domain and their structure (the ARM). These two representations are separate and unrelated. They are initially based on subjective and ad hoc interpretations of the expert's knowledge. Because the representations are new and complex to the domain experts, they are not easily checked and require many cycles of iteration to converge to a meaningful result. Later the ARM is elaborated and translated by the modeling experts to a full product model (or AIM) based on the ISO STEP integrated generic resources (IRs)[2]. The IRs define "a generic ontology for product data and provide the context of the AP domain ontology (Danner 1997)." The product modeling process is iterative and converging between the modeling and domain experts. It typically takes at least five years to complete the specialization and approval of a product model. Some efforts have taken more than ten years. Throughout later stages, application developers within the domain are engaged and translators to/from the product model are developed. A product model specification is implementation-free; it can be mapped into a text file format, an XML Document Object Model (DOM) or an XML schema, a relational or object-oriented database schema, or object model direct mapping interfaces. Initial interfaces typically involve file-based exchange, with database implementations following. The STEP method using IDEF0 has been adopted by many organizations such as US Air Force, IAI, and a number of projects carried out under the

[2] ISO STEP Parts 41 to 56 define IRs.

auspices of the European Union (CIMSteel Integration Standards Release 2 2002; Karstila 2001).

Table 1.1 Mapping between the STEP models and the three-level database architecture

Layer	Model	Languages	STEP Model
External or view level	External Schema or View	IDEF0 IDEF1x NIAM EXPRESS-G EXPRESS	AAM*, ARM *An AAM is primarily an activity / process model, but also represents information flow in a process at a high level (e.g., IDEF0 ICOM).
Conceptual Level	Conceptual or Logical Schema	EXPRESS	AIM
Internal Level	Internal or Physical Schema (Examples include internal data models of CAx3 and other applications as well as database management systems.)	C++ Java XML (SQL)	STEP only provides an interface (i.e, SDAI) to the physical schema.

While the ISO-STEP methodology has been a significant step forward and has allowed integration to be realized that could not have been achieved by earlier file-format technologies, it suffers from a number of drawbacks:

1) The ISO-STEP product modeling process is a *social process* that involves iterative review processes, rather than a rigorous collecting and processing of strategic information (Eastman, Lee, and Sacks 2002). It relies on intuition, tacit expertise and craftsmanship of the product modeling committee. Product modeling needs to be put on a more rigorous scientific foundation, based on a more formal and thus a systematically improvable process.
2) Current methods rely exclusively on human review for validation. While human review is necessary for capturing semantic fallacies, consistency conditions regarding information use within a process and product model can be identified

[3] Application types starting with the phrase "Computer-Aided (CA)": e.g., Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), and Computer-Aided Engineering (CAE)

(Lee et al, 2002). These define logical propositions supporting automatic validation checking, reducing the range of manual checking required.

3) In almost all industry-wide product modeling efforts, IDEF0 models are built as single unified models to represent idealized industry-wide processes, defined by consensus among multiple stakeholders (Katranuschkov et al. 2002; NIST 1993). In this approach, any company level interest in planning its integration with the product model must be carried out separately from the communal activities. There is no means to include these variations in the modeling effort or to validate that the product model developed supports current or anticipated individual corporate processes.

4) Current product data models are defined as static structures, defined more as archives of data rather than as support for strategic workflow processes. The developmental and evolutionary aspects of product development and production planning are not well supported (Eastman and Fereshetian 1994). If product models are to truly support process re-engineering and integration, closer linkage with the workflow characterization of a product domain is required, to explicitly incorporate the developmental aspects of engineering and design.

1.3 A BASIC APPROACH AND PRIMARY GOALS

Process modeling and *product modeling* are currently two different modeling methods with different purposes for representing a domain. A *process* is a series of activities that are “a piece of work that forms one logical step within a process” (WFMC 1999). On the other hand, a product model describes the definition, structure, and relation

of information required to design, engineer, produce, and manage a product. Product modeling serves information structure analysis, software development, database design, and also organizational knowledge management and learning (Bernstein, Pal, and Shutt 2000). These two different modeling methods are related to each other by *information*. Even though information used in a process is not directly depicted in most process modeling methods, conceptually all the activities require input information to perform their tasks and produce output information. The activity–specific information is closely allied to the task-specific software applications developed to support an industry, so there is a strong correspondence between the activity flows and application-specific data exchange requirements. Since the exchange requirements are precisely the purpose of a product data model, the process model can serve as an excellent source to identify many of the semantic constraints applied in developing a product model.

The primary goal of this study is to develop the logic and procedures supporting a formal method for product modeling, based on process-model-derived data. The basic approach is to interweave (or to map) process modeling with product modeling. It aims to provide a scientific foundation to elicit and collect information and domain knowledge through process modeling that is sufficient to replace more traditional modes of conceptual modeling and to (semi-) automatically derive and normalize a product data model from the collected information (Figure 1.2, b). Some requirements collection and modeling methods such as IDEF0 and DFDs allow users to define input and output information at a high level as shown in Figure 1.1 (ISO TC 184/SC 4 1999) or even at a detail level[4]. However, there

[4] In DFDs (Data Flow Diagrams), detailed information transferred between systems can be specified in a separate data dictionary.

is no logic or procedures yet to automatically derive a product model from the specified input and output information without human intervention.

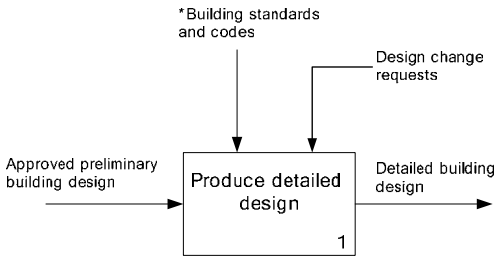


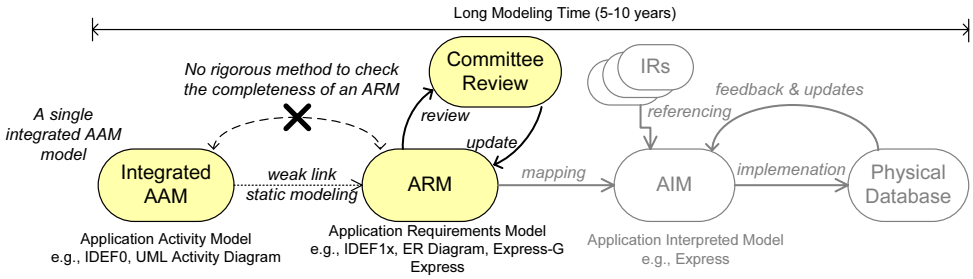
Figure 1.1 A partial IDEF0 model of ISO STEP Part 225

Another goal of this study is to provide the logic to integrate information requirements collected from multiple AAMs into an ARM (Figure 1.2). As discussed earlier, most standard product models today are developed based on a single unified/integrated process model (AAM). And the single unified AAM is used only as a means to define the scope and the context of a product model at a high level. It is not because an AAM method is prohibiting multiple AAM generation or encouraging a single unified AAM development. It is because there has not been a rigorous theory to integrate information requirements specified in multiple AAMs into an ARM and, thus, it is only time-consuming to produce multiple AAMs.

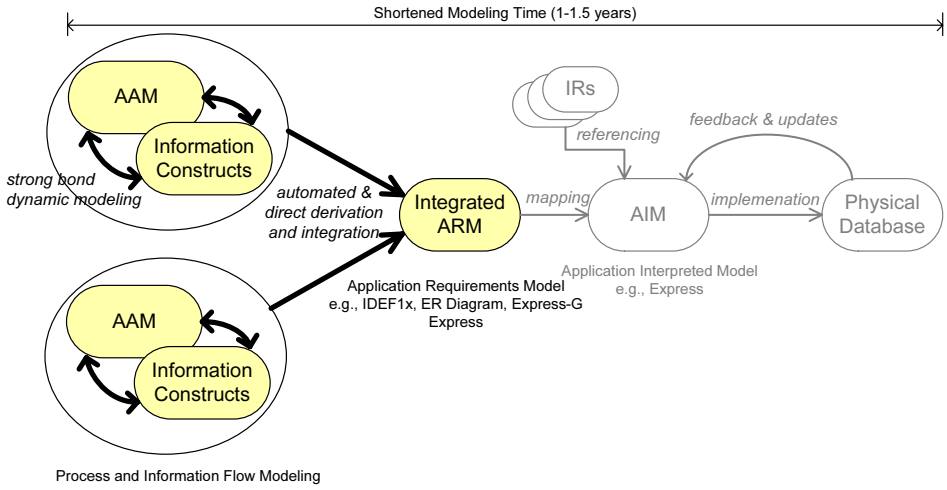
The theoretical goal of this work is to provide a formal structure to the information collection, mapping, and structuring activities that are now used in an ad hoc way in product modeling activities so that product modeling has a more scientific basis, rather than only a social, information standardization basis.

The practical far-reaching goal of this work will be in reducing development time of product models from the current 5 to 10 years to 1.5 years or less by minimizing the committee review cycles, automating the product-modeling processes, and providing a

logical foundation to check the validity. Reducing the development time is essential if product modeling is to facilitate future re-engineering and automation in various industries. It will become more critical as more standard product models are developed to support data sharing between heterogeneous business and application environments. In the future, if a product model cannot satisfy rapidly changing business and software environments, it will become a restriction on design and manufacturing innovation.



a) Traditional Product Modeling Methods



b) A Proposed Product Modeling Method

Figure 1.2 Traditional & proposed product data modeling methods[5]

[5] The diagrams in grey are outside of the scope of this study.

1.4 RESEARCH QUESTIONS AND THE SCOPE

Derivation of a product model from process information is not just a simple process of adding information items to each activity and aggregating them back. First, process information must be constructed as machine-readable information items having a corresponding semantic representation in a product model. And the semantic concepts identified in the process model should be mapped to product data model constructs. Theoretically, the mapping from the captured process information to product data model constructs is similar to the mappings from a data dictionary (a collection of data) to a logical model, and eventually into a physical model. The information items arbitrarily defined in natural language are not adequate for automating the mapping process. Formal methods to define information constructs in a machine-readable format and to incrementally structure the information constructs into a targeted data schema should be provided. In this process, a resultant data schema should be *normalized* (decomposed and restructured) in a logical form. Also appropriate schema integration methods to compose the mapped product model constructs into an overall schema consistent with all the constructs should be developed. While workflow systems[6] have been able to achieve this kind of synthesis for business data, it has not been possible for complex engineering data. These research questions can be summarized as follows:

[6] See Appendix E for a short review on workflow management systems.

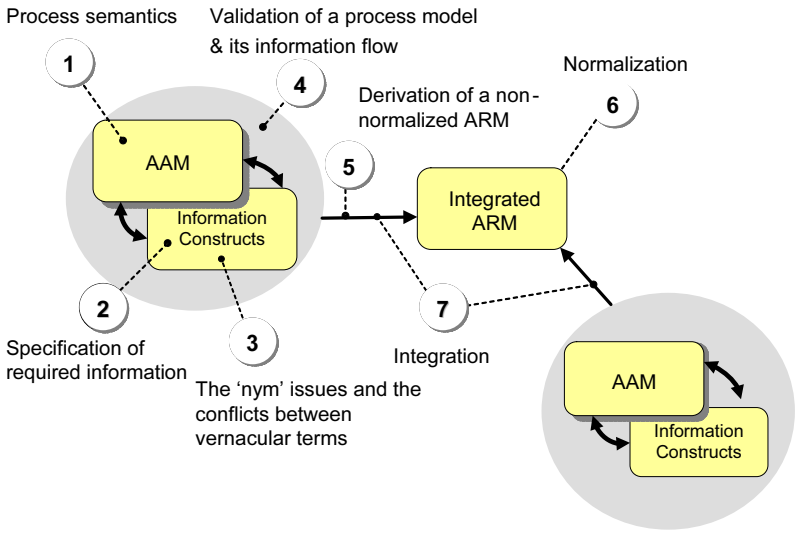


Figure 1.3 Research questions

1) What is the process semantics that is required to elicit processes and information necessary and sufficient to derive a product model?
2) How to specify required information in a machine-readable format
3) How to resolve the '*nym*' issues (e.g., synonyms and homonyms) and the conflicts between company-specific vernacular terms and a consistent machine-readable terms
4) How to validate the consistency of information captured in a process
5) How to derive a product model from the collected process information
6) How to validate the well-formedness of the derived product model and normalize the derived product model
7) How to integrate (or harmonize) product models into one unified model when several different product models are derived from different processes about the same product.

The scope of this study is limited to the development of an integrated ARM. A theoretical foundation for automating the mapping between an ARM and an AIM is outside of the scope of this study.

Chapter 2 briefly reviews the history of product modeling and the existing product models in the Architecture, Engineering, and Construction (AEC) domain[7].

Chapter 3 provides formal definitions of two product modeling approaches: i.e., the application-centric approach and the process-centric approach. It also formally defines the relationship between a process model and a product model.

Chapter 4 introduces the Requirements Collection & Modeling (RCM) phase of the proposed method. It discusses process semantics required for deriving a product model from collected information requirements and describes a grammar for product information using a Context-Free Grammar (CFG). Also it describes the logic for checking the consistency of information flow within a process.

Chapter 5 discusses the Logical Product Modeling (LPM) phase of the proposed method and proposes nine design patterns to integrate and normalize collected information requirements into an ARM.

Chapter 6 explains how the proposed method has been implemented based on an assumed product modeling process.

Chapter 7 reviews and evaluates the method. The proposed method were experimented with fourteen precast producer members in the US and Canada. Three product models and an integrated product model have been automatically generated from collected information requirements through the proposed product modeling process. The

[7] The facility management (FM), real estate, infrastructure industries are often treated as separate industries from the AEC industry. However, this paper uses “the AEC industry” as a term, which also includes all other relevant industries.

results were compared with a data schema of an existing ERP system and with the precast concrete extension of an existing standard product model (i.e., PCC-IFC).

1.5 GLOSSARY

- activity: a logical step within a process (WFMC 1999) (Section 1.3) From a product-modeling point of view, an activity of a process can be defined as *an act of processing information items* (Section 3.4)
- application activity model (AAM): 1) the engineering process context in which an AP will be used (VTT Building and Transport 2002)" (Section 1.2); 2) a model that describes an application in terms of its processes and information flow (ISO JTC 1/SC 32 2003)
- application context: the intended use of product data within an application (ISO JTC 1/SC 32 2003)
- application interpreted model (AIM): 1) a logical model of the information domain (Section 1.2); 2) an information model that uses the *integrated resources* necessary to satisfy the information requirements and constraints of an *application reference model* (ISO JTC 1/SC 32 2003)
- application protocol (AP): 1) The STEP name for a product model of each domain is an Application Protocol (AP) (Section 1.2); 2) a part of the ISO STEP standard that describes the use of integrated resources satisfying the scope and information requirements for a specific *application context*. (ISO JTC 1/SC 32 2003)
- application reference model (ARM): 1) a view of the information domain (called an Application Requirements Model (ARM)) (Section 1.2); 2) an

information model that describes the information requirements and constraints of a specific application (ISO JTC 1/SC 32 2003)

- application: a group of one or more processes creating or using product data (ISO JTC 1/SC 32 2003)
- flow: relation (e.g., transition) between activities. (Section 3.4)
- Georgia Tech process to product modeling (GTPPM): the process-centric product modeling approach, which consists of the *Requirements Collection and Modeling* (RCM) module and the *Logical Product Modeling* (LPM) module (Section 3.5)
- information construct (IC): 1) a formally defined information item used within a process. (Section 3.5); 2) a concatenation of tokens, which conforms to the product information specification (PIS) grammar (Section 4.6.1)
- information item: a minimum expression of product information. (Section 3.4)
- information menu (IM): 1) a collection of tokens possibly used in a UoD with a classification structure. It restricts the ways in which tokens can be strung together in constructing information item. (Section 3.5); 2) a collection of tokens that forms a minimum expression (or phrase) of product information (Section 4.6.1)
- information unit: a grouping of relating constructs (entity data types, attributes and relationships) that together represent one of the high level concepts of the STEP data architecture (Fowler 1996)
- integrated resource: a part of the ISO STEP standard that defines a group of *resource constructs* used as the basis for product data (ISO JTC 1/SC 32 2003)

- logical product modeling (LPM): an algorithmic process to derive a product model from collected information constructs (Section 3.5)
- model: an abstract representation or description (ISO JTC 1/SC 32 2003)
- normalization: 1) an activity of using the known semantics of data in the form of dependencies that may be a cause for potential “update anomalies” requiring unnecessary duplicate work as well as causing potential inconsistencies in a database. (Section 5.2); 2) decomposition and restructuring of a data structure to a normal form (Section 5.4)
- production information specification (PIS) method/mechanism: a method to specify product information in a consistent, extensible, generative, analyzable, and accessible manner (Section 4.6)
- process model: a model that describes how activities within a process are connected, ordered, and structured, and represents a *use case* of information. (Section 3.3)
- process: a series of activities
- product data: a representation of facts concepts, or instructions about a product or set of products in a formal manner suitable for communication, interpretation, or processing by human beings or by automatic means (ISO JTC 1/SC 32 2003)
- product information: 1) the information generated, used, and maintained throughout a product's lifecycle. (Section 4.6) 2) facts, concepts, or instructions about a product or set of products (ISO JTC 1/SC 32 2003)
- product model (or product data model): 1) a formally structured schema of such product information that is generated, modified, and deleted throughout a

product's lifecycle (Section 1.1); 2) a model that describes the definition, structure, and relation of information required to design, engineer, produce, and manage a product. (Section 1.3)

- product: 1) a thing or substance produced by a natural or artificial process (ISO JTC 1/SC 32 2003); 2) the identification and description, in an *application context*, of a physically realizable object that is produced by a process (Fowler 1996)
- requirement collection & modeling (RCM): a graphical Requirements-Collection-and-Modeling language for capturing information in the context of its use (Section 3.5)
- resource construct: the collection of EXPRESS language entities, types, functions, rules, and references that together define a valid description of product data (ISO JTC 1/SC 32 2003)
- semantic intersection: a set of information items in two different data sets that is semantically equivalent. (Section 3.2)
- state: A state (S) is a mode of a project. The state of a project is changed by a set of activities (A). A project cannot autonomously change its state. (Section 3.4)
- supertype: a set of least common attributes of its subtypes (Section 4.6.1)
- token: a non-decomposable meaningful lexical element (ISO TC 184/SC 4 1994) (Section 3.5) (Section 4.6.1)

- vernacular data dictionary (VDD): a data dictionary of vernacular information items (VIIs), which includes VII names, definitions, data type, examples, references, and synonyms (Section 6.2)
- vernacular information item (VII): a company-specific local nomenclature and definition for product information (Section 3.5)
- view: a *semantic subset* of its superset similar to the concept of semantic intersection; a *derivable subset* from its superset. (Section 3.5)

CHAPTER 2

BACKGROUND

2.1 OVERVIEW

This chapter discusses why a standard product models is required and briefly reviews the early product modeling efforts and product models in Architecture, engineering, and construction (AEC).

2.2 NEEDS FOR STANDARD PRODUCT MODELS

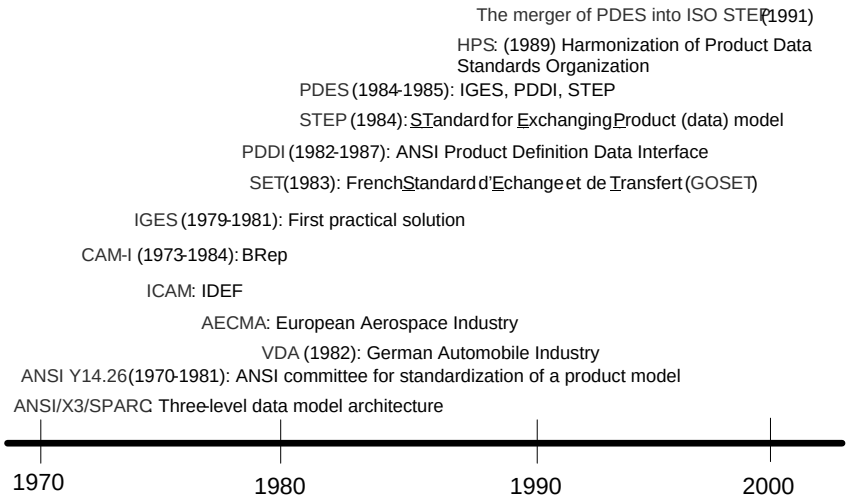


Figure 2.1 A timeline of product modeling efforts

The (standard) product modeling efforts first began as an effort to exchange a set of geometric data between different CAD systems in the 1970s. Even at that time, when there were only a few CAD systems with any significant market penetration, the demands for standard geometry and topology to exchange data between different CAD systems were

very well recognized (Goldstein, Kemmerer, and Parks 1998). Over the time, the scope of product information, which can be managed electronically, has been broadened and so does that of product models and the number and types of software applications. Figure 2.1 is a timeline of those product modeling efforts. A brief summary of each project is provided in Appendix A. Detailed and good descriptions on each project are available in (Bloor and Owen 1995; Eastman 1999; Goldstein, Kemmerer, and Parks 1998).

Figure 2.2 is a well known diagram that illustrates the needs of a standard product model in terms of the number of translators required for exchanging data between *n* numbers of software applications with and without a standard product model. Figure 2.2 (a) illustrates a case where there are *n* numbers of applications but without a standard product model and Figure 2.2 (b) a case where there is a standard product model.

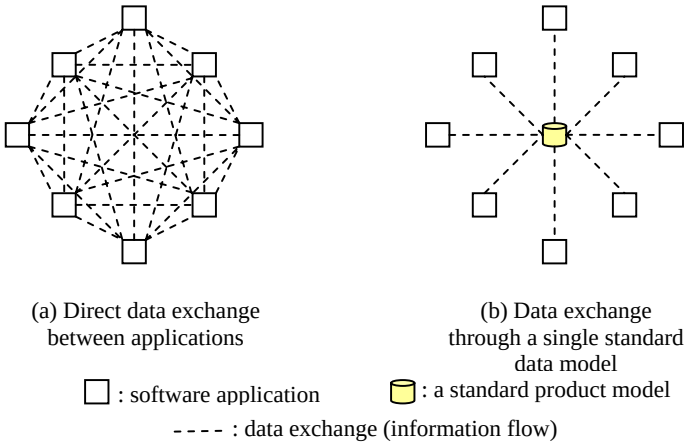


Figure 2.2. Data exchange between different applications

Each application needs at least two translators to import and export data to another application in both cases. The number of translators required for exchanging data between applications in Case (a) is $2 * n * (n - 1)$ or $2n^2 - 2n$ and in Case (b) $2n$. The difference increases exponentially as the number of applications increases. Since more and more

software applications with various functions and formats are released to the market every year, the standard data model approach seems very cost-effective and time-saving compared to the direct data exchange approach. However, this comparison has been criticized for being too idealistic. Some of the criticisms[8] are as follows: (See Figure 2.3 for an example)

- A company or a project does not use all the software applications available in the market (Figure 2.3), but only a small subset of the software applications available in the market (Figure 2.3).
- Not all the software applications used by a company need to talk to each other. For example, usually there is no data exchange between a CNC machine and a structural analysis system (Figure 2.3).
- Through the last twenty or thirty years, software applications became versatile. One application or a bundle of applications by one software vendor can support the broad range of product design, engineering, and production activities.
- Some applications have embedded direct links between themselves and different applications developed through Application Programming Interfaces (APIs) (e.g., a CAD system and a structural analysis system in Figure 2.3). Some relevant technologies are the middleware (e.g., ODBC), the Dynamic Link Library (DLL), and the Component Object Model (COM) technologies.

[8] This criticism is based on a survey on the use of software applications in the precast concrete industry, interviews with architects, discussions with software developers, Fried Augenbroe's presentation at ECPPM 2002 (Augenbroe 2002). Another set of discussions on a standard product model can be found in (Amor 2001). Rober Amor discussed twelve common misconceptions (or misbelieves) about standard product models and integrated project databases. Those are: 1) OO provides the complete solution; 2) The single data model will appear; 3) We represent reality; 4) User views are reconcilable; 5) Mapping is easy; 6) The Internet solves the communication problem; 7) XML solves the representation problem; 8) Documents will disappear; 9) CAD is the center of an integrated project database (IPDB); 10) IPDB solves information ownership problems; 11) IPDBs guarantee coordinated and consistent information; and 12) The industry is ready for IPDBs.

- Some applications are dominant in a certain domain (e.g., AutoCAD in AEC). And their data formats are often used as *de facto* standard data models for certain types of applications (e.g., DWG or DXF). They are limited in many ways, but still usable.
- Even if there is a standard data model, only a selected set of data can be exported or imported between different types of applications. For example, usually an Enterprise Resource Planning (ERP) system may not read in all the geometric data from a Computer Aided Design (CAD) system and a CAD system will not read in managerial data from an ERP system.
- Sometimes unidirectional data exchange is preferred by companies. For example, many architectural firms are very reluctant to give electronic copies of their projects to third parties unless they have a strong business relationship or are forced to share information by building codes because 1) they do not want reveal their business secrets and design esoterics; 2) there are always potential legal issues; and 3) technology is not there yet: e.g., the exchange process often loses or alters data. And there is no rigorous method to keep track of changes or to validate an exchanged model yet. For this reason, many AEC companies today do not read in an electronic model from another party as it is, but rather incorporate changes into their own model one by one manually.
- Also many software vendors are not willing to make their applications interoperable because they believe that they will lose competitiveness in the market by supporting data exchange between theirs and other applications (Szykman et al. 2001).

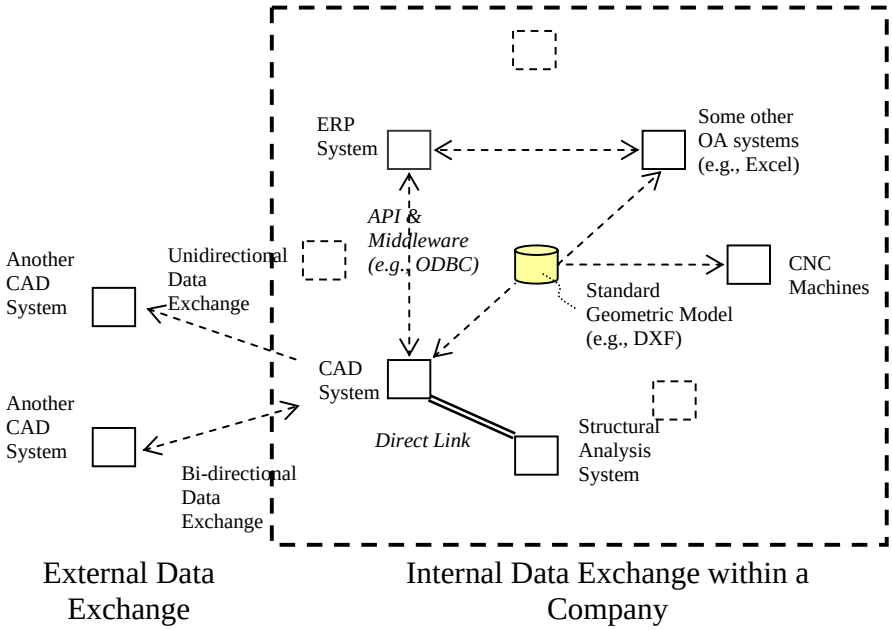


Figure 2.3. Internal and external data exchange in practice

Although the benefits of a standard product model are not as great as they are in an ideal situation, there are still several reasons to develop standard product models:

- First, different projects or companies use different sets of software applications. Thus, software vendors need to support not one set of applications as shown in Figure 2.3, but multiple sets of data exchange scenarios. Not all the applications need to talk to each other as shown in Figure 2.2, but the exchange scenario can still be pretty complex as reported in (Fischer and Kam 2002).
- The above argument is more true to the AEC industry than to the manufacturing industries (including the automobile and the aerospace industries) because, unlike them, companies in the AEC industry work like a temporary consortium a project by a project (more like the movie industry) or a region by a region. There can be a fixed set of software applications within a company, but not across companies in the AEC industry.

- Each application has a proprietary internal data structure. Even though many software applications provide an open Application Programming Interface (API) today, it is still not an easy job to understand the internal data structures of all the targeted applications and develop and update translators between them.
- Software applications and their internal data structures are usually updated every year or two. Even if a software application supports data import/export functions only for a small number of applications, it will be time-consuming and expensive to update translators every year.
- Some software vendors do not want to reveal the internal data structure of their applications. In such cases, the translator development entails code-hacking and can possibly lead to a legal dispute as a result.
- As the interest in the concept of a *central product model repository (PMR)*[9] as a means of *product/project lifecycle management (PLM)* and as a substitute for file-based data exchange issues (You 2003) increases, the importance of a standard product model especially in a collaborative work environment has further emphasized by many studies (Adachi 2002; Amor 2001; Augenbroe 2002; Hardwick et al. 2000; You, Yang, and Eastman 2004).

Industries, in fact, squander billions of dollars due to poor interoperability between software applications (Szykman et al. 2001). A standard product model is an open public data schema and can eliminate or reduce most of the issues described above. However, there will be still many other technical and cultural issues in interoperability (e.g., the

[9] a.k.a. an Integrated Project DataBase (IPDB,(Amor 2001)) and a Virtual Enterprise Product data Repository (VEPR, (Hardwick et al. 2000)) .

concurrent engineering issues; the change propagation and management issues) that a standard product model cannot resolve. In any case, if a standard product model cannot be delivered to the software developers in time, all these discussions are meaningless even in the first place. An efficient and scientific product modeling method, which can generate a rigorous and practical product model in a short period of time, is critical in the success of the standard product modeling effort. This study aims to develop such a product modeling method.

2.3 EARLY BUILDING PRODUCT MODELS

There have been many efforts to develop building product models. Early *building product models* include Jim Turner's Building System Model (BSM) (Turner 1988, 1988), Gielingh's General AEC Reference Model (GARM) (Gielingh 1988, 1988), the Finnish RATAS project (Bjork 1989), and the Construction Integrated Manufacturing for Steel Structures (CIMsteel or CIS)(AISC 2002; EUREKA 1987-1997).

Wim Gielingh was the chairman of the ISO-STEP AEC committee at that time and both the BSM and the GARM were working STEP documents. The subcommittee was called TC184/SC4 WG1[10]. The BSM decomposed a building project into a single site, a building, and a collection of (sub-) systems (Turner 1988). It used NIAM as a modeling language. An interesting aspect of the BSM is that it initially proposed, so called, a "shotgun" approach: i.e., exchanging data through generic OBJECT, ATTRIBUTE, and VALUE objects (Figure 2.4[11]) (Turner 1988) instead of exchanging data through building-industry specific objects and attributes (e.g., an object DOOR has attributes MATERIAL,

[10] TC: Technical Committee, SC: Sub-Committee, WG: Working-Group

[11] A model in NIAM is provided in Appendix B.

COLOR, STYLE). In a sense, this approach is similar to the *late binding* approach in computer programming. But this approach does not work especially for exchanging data between object-based CAx systems because there is no guideline to determine what information means what: e.g., 'tread_width' in one system can mean 'tread_length' or 'tread_depth' in other systems. In order to avoid any misinterpretation, there should be a separate standard data model to define the domain-specific objects and attributes.

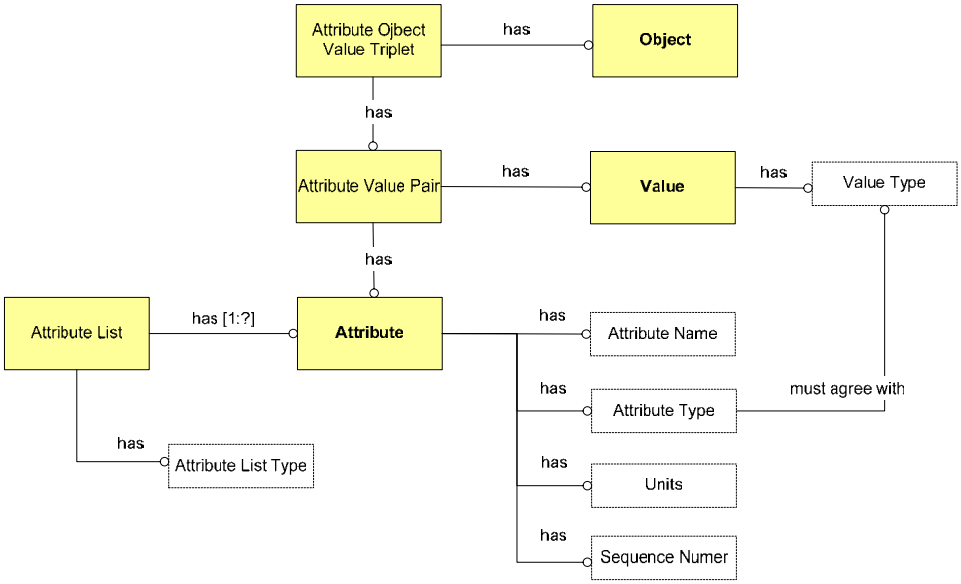


Figure 2.4 The attribute properties model of the Building System Model in EXPRESS

GARM was initially proposed as a generic data model to integrate various models developed within AEC and other models in STEP/PDES (Gielingh 1988). It included the Product Definition Unit (PDU) entity and several subtypes (e.g., the Functional Unit (FU) entity and the Technical Solution (TS) entity) (Figure 2.5). PDUs in AEC are Building, Plants, Ships, and Civil Engineering. GARM does not predefine what a PDU is: it can be a system, a sub-system, a component, a part, a feature, a space, or a joint. A Function Unit (FU) represents a requirement for a PDU. A Technical Solution (TS) is an answer to the

requirement. Such relations between FUs and TSs are described in, so called, a *hamburger diagram*. Figure 2.6 illustrates an example of the hamburger diagram.

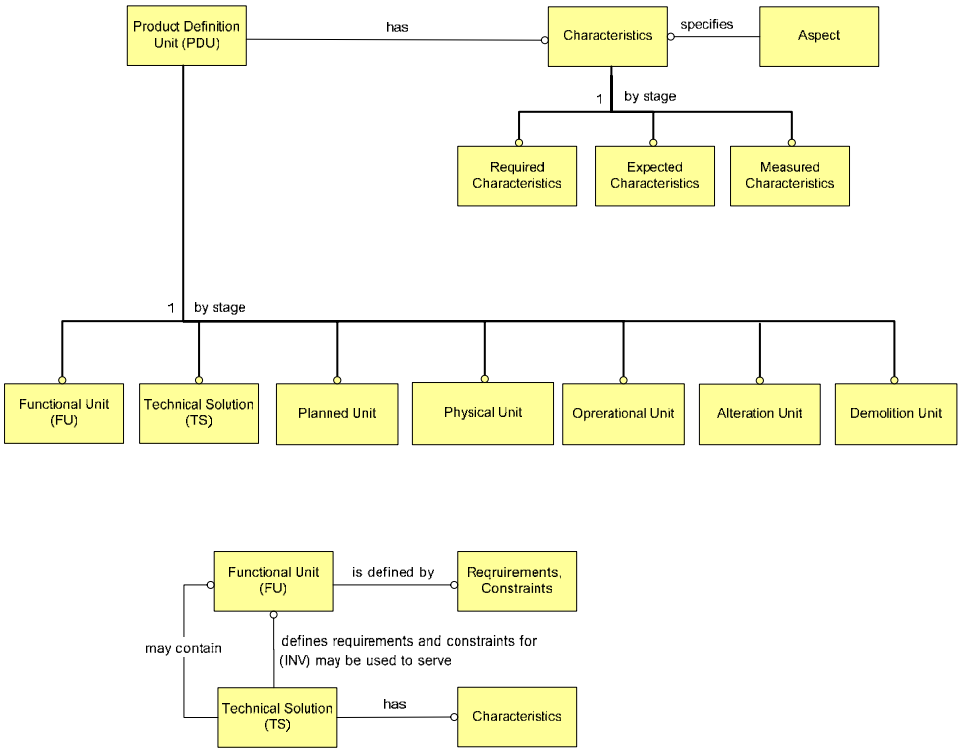


Figure 2.5 The PDU entity and its subtypes in the GARM

The GARM and the BSM were followed by the Building Construction Core Model (BCCM) ISO STEP Part 106 by Jeffrey Wix in 1994. The BCCM was regarded as a framework model and lacked detailed definitions of objects (Eastman 1999). It was later withdrawn from the ISO STEP Integrated-application Resources (IR) list.

The RATAS project was led by Bo-Christer Bjork at VTT in Finland (Bjork 1989). RATAS categorized a building into five levels: building, system, sub-assembly, part, and detail. One of interesting aspects of the RATAS model is that it categorizes SPACE and JOINT as an individual entity, not as an attribute or a relation (Figure 2.7) (Eastman 1999).

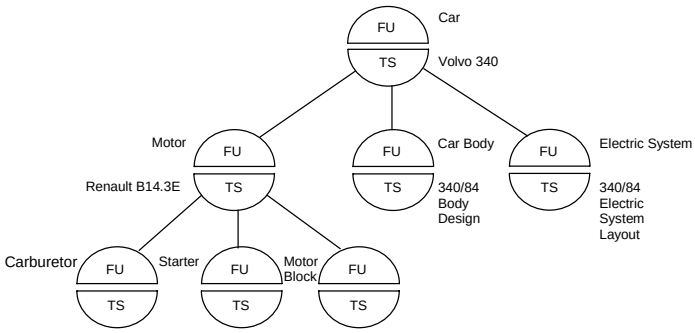


Figure 2.6 A hamburger diagram

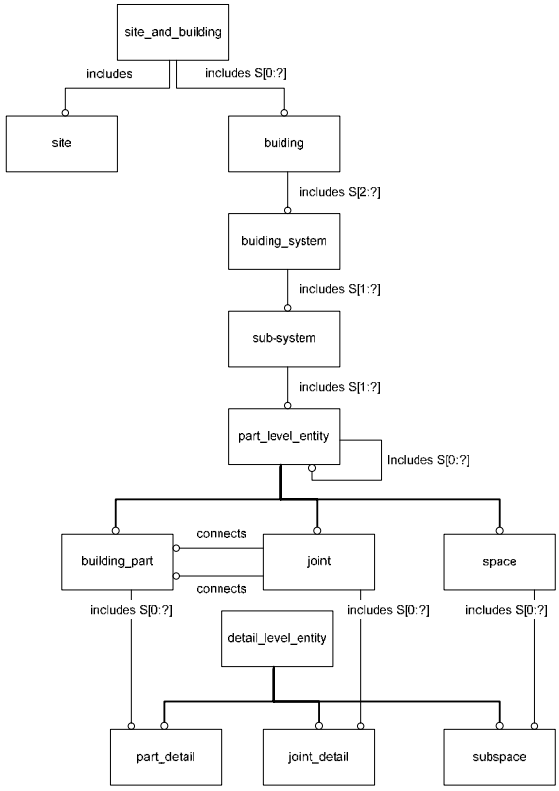


Figure 2.7 The RATAS building kernel model, defined as an abstraction hierarchy

These models were framework models and have not been broadly accepted by software vendors. On the other hand, the CIMsteel (CIS in short)(Crowley 2000; Crowley and Ward 1999) and the Industry Foundation Classes (IFC) (IAI) models are the only two

models that are practically and widely deployed by the AEC industry for exchanging data today. These two models are compared and reviewed in the next section.

2.4 CIS AND IFC

The CIS (CIMsteel) model was initially developed by Andrew Crowley and Alastair Watson at the University of Leeds (Crowley 1998) as part of the EU EUREKA project (EUREKA 1987-1997). The current version of the CIS model is CIS/2 LPM6 and is still maintained by Andrew Crowley supported by the American Institute of Steel Construction (AISC).

The IFC model has been developed and maintained by the International Alliance for Interoperability (IAI) since 1994. The current version of IFC is IFC2x2. And there are thirteen completed extension projects and seven ongoing extension projects as of March 30, 2004. A short history of the IAI and the IFC is available at (IAI 2004).

The commonality between the CIS and IFC models is in that both of them are industry-driven efforts even though the CIS project was initially an academe-led project with support from a large industry team. The success of both models may be attributed by this industry-level support. Currently there are nineteen software companies including AutoDesk, Bentley, and Graphisoft that are involved in the IFC projects (IAI) in the North America. And twelve software companies including Tekla, Intergraph, and Bentley are involved in the CIS project (Yang et al.). Nevertheless, while IFC2x2 is adapted still in a limited manner in real projects, the American Institute of Steel Construction (AISC) informally reported that over 50% of the AISC steel fabricators is exchanging data using CIS/2. A clear reason that IFC2x2 is deployed only in the limited scope of a project is that it still lacks detailed object definitions, which are essential for exchanging information of

real construction projects. This difference is due to IFC's and CIS' different goals, scopes, modeling approaches, and styles. First, the goal of IFC is to develop a core high-level model to which AEC-specific extensions can be added later (IAI 2000). IAI explains that, if there is a huge model that contains all the information in the AEC, the model would be "high complex and difficult to understand and virtually impossible implement." The domain-specific definitions are assumed to be added as a "leaf node (extension)" to the core IFC model. Currently IAI is supporting many extension modeling efforts. The current and completed IFC extension projects are listed in Table 2.1 as of March 30, 2004.

Thus, the structure of the IFC model is conceptual and generic. The backbone entities of IFC2x2 stems from the `IfcRoot` entity. `IfcRoot` is subcategorized into three conceptual entities: `IfcObject, IfcPropertyDefinition`, and `IfcRelationship` similar to the basic three components of the Relational database approach: i.e., Entity, Property (Attribute), and Relation:

```
ENTITY IfcRoot
 ABSTRACT SUPERTYPE OF (ONEOF
        (IfcObject
        ,IfcPropertyDefinition
        ,IfcRelationship));
        GlobalId : IfcGloballyUniqueId;
        OwnerHistory : IfcOwnerHistory;
        Name : OPTIONAL IfcLabel;
        Description : OPTIONAL IfcText;
 UNIQUE
        UR1 : GlobalId;
END_ENTITY;
```

On the other hand, the CIS model targeted a very specific domain (i.e., the steel construction industry) and is structured according to four high-level processes in the

construction steel industry[12]: Design, Analyze, Return Analysis Results, Modify Design, and Manufacture. This process is described in detail in (Crowley and Ward 1999) as an IDEF0 model. Information used in the four processes are modeled as four subset models called the analysis model, the analysis result model, the design model, and the manufacturing model accordingly. The distinction between these subset models has been blurred while the conflicts between models were resolved through updates. However, the initial modeling philosophy is still well integrated into the current model.

Beyond the overall structure, the CIS and the IFC models have minor differences. In terms of a modeling style, the CIS model uses the ANDOR constraint, which causes many problems in implementation, while the IFC model excludes the ANDOR constraint (IAI). Entities in the IFC model are all named starting with "Ifc", which makes reading and sorting of entity names difficult.

Table 2.1 IFC extension projects

Completed Projects	**Ongoing Projects**
1) HVAC Performance Validation [BS-7*] 2) HVAC Modeling and Simulation [BS-8] 3) Network IFC: IFC for Cable Networks in Buildings [BS-9] 4) Code Compliance Support [CS-4] 5) Electrical Installations in Buildings [EL-1] 6) Engineering Maintenance [FM-1] 7) Costs, Accounts and Financial Elements [FM-8] 8) Material Selection, Specification and Procurement [PM-3] 9) Steel Frame Constructions [ST-1] 10) Reinforced concrete structures and foundation structures [ST-2] 11) Precast Concrete Construction (PCC)** [ST-3] 12) Structural Analysis Model and Steel Constructions [ST-4] 13) IFC drafting extension [XM-4]	1) Early Design [AR-5] 2) Bridge [CI-2] 3) Industry Foundation Classes for GIS (IFG) [CI-3] 4) Electrical Installations in Buildings (EL-2) [EL-2] 5) Portfolio and Asset Management - Performance Requirements (PAMPeR) [FM-9] 6) Structural Timber Model [ST-5] 7) Harmonization of ISO 12006 Part 3 with IFC [XM-7]
* The numbers in parenthesis are extension identifier numbers ** The ST-3 project is also known as the PCC-IFC project.	

[12] It is also possible to say that the CIS model is modeled depending on four different application functions.

Every model has a different style depending on its purpose and assumptions. IFC and CIS also have different styles based on their different goals. Thus, it might not be valid to judge which one is better over another. However, the method, which this study aims to develop, should be able to allow various data modeling style. This issue is discussed in detail in Section 4.6.4.

2.5 OTHER BUILDING PRODUCT MODELS & RELEVANT PROJECTS

Figure 2.8 summarizes major product modeling efforts in AEC. As shown in Table 2.1, IFC recently added the cast-in place (CIP) concrete extension (ST-2), the precast concrete extension (ST-3) (Karstila et al. 2002) and the construction steel extensions (ST-1 and ST-4). Since these were all driven by the European Union, the resultant models do not satisfy some of the demands of the North American AEC industries. In parallel to these efforts, Chuck Eastman at Georgia Tech is leading a project to develop a product model for the North American precast concrete industry for the last three years. The model is tentatively called a *Precast Concrete Product Model (PCPM)*. It is clear that the mapping and harmonization between product models will be a critical issue in the near future. And there is already a movement to respond to such issues.

Other building product models and relevant projects include:

- Building Elements (1994) Wolfgang Haas, STEP Part 225;
- BSAB (Ekholm 1996; Ekholm and Fridquist 1996);
- Building Lifecycle Interoperable Software (BLIS, http://www.blis-project.org);

- Architecture, Methodology and Tools for Computer-Integrated Large-Scale Engineering (ATLAS) (Tolman and Poyet 1995);
- Virtual Enterprise using Groupware tools and distributed Architecture (VEGA); VERA at VTT (1997-2002);
- Computer Models for the Building Industry in Europe (COMBINE I & II)(Augenbroe 1993, 1995);
- the Engineering Database Model (EDM) project (Eastman, Chase, and Assal 1993);
- the Intelligent Services and Tools for Concurrent Engineering (ISTforCE) project (Wix and Liebich 2000);
- OSMOS IST-1999-10491 (Wilson et al. 2001);
- Electronic Business in the Building and Construction IST-1999-10303 (E-Construct).

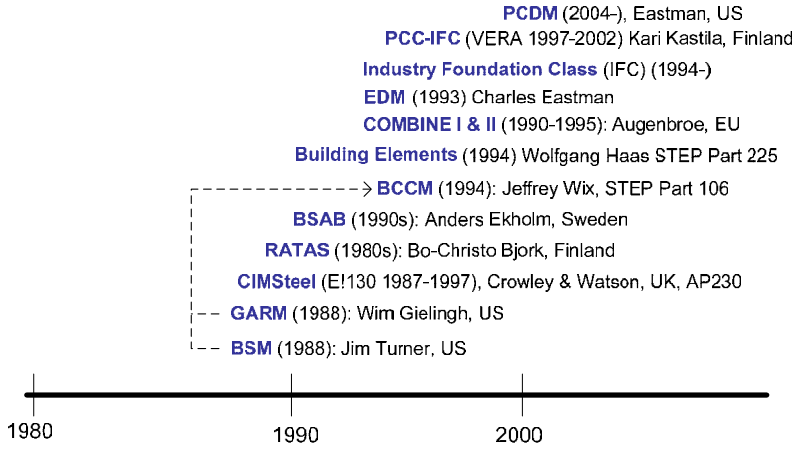


Figure 2.8 A timeline of major product modeling efforts in AEC

Summaries and reviews on some of these models and projects are available in (Christiansson and Karlsson 1988; CSTB 2004; Eastman 1999; Ronneblad 2003). Among

these, the EDM project (Eastman and Jeng 1999) was unique in that it attempted to develop an *evolvable* product model through the lifecycle of a product instead of defining a static product model that can only support the predefined scope of product information. Many advanced engineering database issues such as *incremental schema evolution, concurrent engineering, selective updates,* and *integrity maintenance* were identified and discussed through the project. As a result, a data model and implementation language EDM-2 has been developed and a small case has been implemented on top of UniSQL®. However, the project has been discontinued and the approach has not been rigorously evaluated yet.

2.6 OTHER STUDIES ON PRODUCT MODELING

Much of the literature in product models involves case studies on their application and expected benefits (Giannini et al. 2002; Smith 2002; Szykman et al. 2001). Others focus on new developments of product models that extend their use and support new engineering applications, such as the development of product catalogs (Peak, 2001), support for feature-based design (Dereli and Filiz 2002), and made-to-order products data exchange using parametric models (ISO TC 184/SC 4 2001). In addition, there have been some efforts to define a common set of abstract concepts and relations for product models (Bjork 1989; Eckholm and Fridquist 1996), especially based on function-structure-behavior trichotomy (Fenves 2001). Work has begun to address the prescriptive definition of a product model with linkages to a process model, so that the interactive effects of design changes on processes can be better identified (Feng and Song 2000). Other work has used STEP-models to identify product groups (El-Mehalawi and Miller 2001).

Work has also focused on development of extensions to the basic STEP methods. These include languages for mapping between EXPRESS models (Spooner and Hardwick 1997) and the development of incremental evolution of EXPRESS models (Kahn et al. 2001) and analysis of abstraction level (Mannisto et al, 2001). An effort somewhat related to this study was to develop an EXPRESS product specification schema (McKay, de Pennington, and Baxter 2001). This work builds upon product specification concepts of (Pahl and Bietz 1998) to capture the requirements for made-to-order products. The requirements are for the product, however, not a product model.

CHAPTER 3

A NEW AND FORMAL PROCESS-CENTRIC PRODUCT MODELING APPROACH

3.1 TWO APPROACHES TO DEVELOP A PRODUCT MODEL FOR DATA EXCHANGE

All product models are developed through a *conceptual thinking process*. *Modeling by decomposition* is a good example of conceptual modeling: e.g., A BUILIDNG consists of SUBSYTEMs. A SUBSYSTEM consists of building PARTs. A PART consists of SUBPARTs and so on. But, if a product model is to be developed only depending on a conceptual thinking process, there will be no constraint or reference to determine the scope of a product model. Also, there might be a gap between the resultant product model and actual user requirements. Thus, a product model should be defined in a certain context or within a specific scope.

The scope or context of a product model for data exchange can be defined generally by two ways: i.e., by native data structures of *software applications* of interests or by *activities and processes* of interests. These approaches can be respectively called an *application-centric* approach and a *process-centric* approach. Since applications also operate to support a process, these two approaches are not mutually exclusive. However, these two approaches are taking theoretically different approaches to automate/rationalize data modeling processes. The following two subsections formally define and compare these two approaches introducing new *semantic set operations*. The last section of this

chapter introduces and overviews the architecture of a new and formal process-centric product modeling approach proposed in this thesis.

3.2 THE APPLICATION-CENTRIC MODELING APPROACH

Not all the data in two applications can be exchanged. But more than a mathematical intersection of the two native data sets can be exchanged. We call the set of data, which can be exchanged between two data models, a *semantic intersection*. A *semantic intersection* is a set of information items in two different data sets that is semantically equivalent. For example, let's assume that *A* is a set of information required by a delivery management system or corresponding process, and that *B* is a set of information required by a structural analysis system or corresponding process.

$A \equiv$ {project_name, load, driver}

$B \equiv$ {strucutre_name, load, frame}

The results of regular set operations[13] of these two sets will be:

$A + B \equiv$ {project_name, structure_name, load, load, driver, frame }

$A \cap B \equiv$ {load}

$A \cup B \equiv$ {project_name, load, driver, structure_name, frame }

However, it is very unlikely that data models of two different applications use the same terms or the same data structure to define their native data structure. Thus, let us assume that project_name in Set *A* is a *synonym* of structure_name in Set *B* and that load

[13] The regular set operations assume that there is no homonym and synonym in any set.

("truck load") in Set *A* is a *homonym* of load ("structural load") in Set *B*. In such a case, the results of the *semantic set operations* of these two sets will be:

Let $\cap^+$: a set (or aggregation) of semantically equivalent entities

$\cap^*$: semantic intersection

$f_{si}(x, y)$: a function, which returns either one of semantically equivalent information items *x* or *y*; *x* and *y* can be also expressed in terms of functions: e.g., *f(x)* and *f(y)*

$A \cap^+ B \equiv$ {project_name, structure_name}

$A \cap^* B \equiv \{ f_{si}$(project_name, structure_name)}

If f_{si}(project_name, structure_name) = project_name,

$A \cap^* B \equiv$ {project_name}

(The definition and an example of the *semantic union* ($\cup^*$) are provided in Appendix B.)

In this case, only project_name and structure_name can be exchanged between two systems. Others will be lost in the data exchange process. The definition of semantically equivalent items is not limited to synonyms. The *entities in driving and driven relations* can be also regarded as semantically equivalent items. For example, a CAD system usually does not carry "surface_area" in a native data model because the surface area of a shape can be calculated based on other geometric information. On the other hand, an estimation

system often includes “product_surface_area”, but does not manage detailed geometric information of a product. For example, let’s assume that we are interested in “wall_surface_area”. Let A be a set of information in a CAD model. Let B be a set of information in an estimation system. “→” denotes a functional dependency. $A \rightarrow B$[14] denotes “if *A* then *B*” or “*B* is derived from *A*”.

$A \equiv$ {wall_width, wall_height}

$B \equiv$ {wall_surface_area}

(wall_width, wall_height) → (wall_surface_area)

$A \cap^* B \equiv \{ f_{si}(\text{wall\_width} \times \text{wall\_height}, \text{wall\_surface\_area})\}$

$\equiv \{ f_{si}(\text{wall\_surface\_area}, \text{wall\_surface\_area})\}$

In general, if there are driving and driven items, driven items should be regarded as a semantic intersection of driving and driven items because it is usually possible to derive driven items from driving items, but not *vice versa*. Therefore, the semantic intersection of Applications *A* and *B* in the above example is:

$A \cap^* B \equiv$ {wall_surface_area}

However, if the relationship between items is bidirectional (i.e., an item can be both a driving and a driven item of the other item at the same time), all the items should be included.

If $a \in A$, $b \in B$, a→b, b→a, then

$A \cap^* B \equiv$ {a, b}

[14] The same symbol is used in a later section to show a rewrite rule in the Context-Free Grammar.

e.g.,

$A \equiv$ {wall_width, wall_height}

$B \equiv$ {wall_height, wall_surface_area}

$A \cap^* B \equiv$ {wall_height, f_{si}(wall_width $\times$ wall_height, wall_surface_area),

f_{si}(wall_width, wall_surface_area $\div$ wall_height)}

$\equiv$ {wall_height, wall_surface_area, wall_width}

In many cases these relations are not apparent and are difficult to define. (Stouffs, Krishnamurti, and Eastman 1996) is a good example of showing the complexity of mapping different solid representations.

This definition implies two apparent, yet important facts about data exchange between two systems:

1) Theoretically as well as practically, there cannot be lossless data exchange between two applications.
2) The more similar two application types are, the more information they can exchange.

If there are more than two applications, a product model will be the *grand union of all the semantic intersection* of all the applications:

Let A_i and A_j : an application

n: the number of applications

Product Model D $\equiv \bigcup (\sum_{i=1}^{n} A_i \cap^* \sum_{j=1}^{n} A_j)$

This definition is important because it provides an algorithmic definition of a product model and opens up a possibility of automating the development of a translator or a data model: i.e., theoretically, if a *semantic intersection* of all the native data models of interest can be identified, a product model to support data exchange between the native data models can be automatically derived from the identified semantic intersection. Identification of a semantic intersection of two data models basically undertakes the same process as *schema mapping*. As Robert Amor pointed out (Amor 2001), mapping is not easy and there is much work to be done to make automated translator or product model development possible.

However, the application-centric product modeling approach also has several drawbacks. A product model often includes non-existing software applications that users wish to include in their data exchange scenario in the near future. But, based on the above definition, a product model cannot be defined if the data structures of targeted software applications are not predetermined. Also a product model can be used as a standard data schema not only for data exchange between different applications, but also for a central project/product management system (PMS) to support a collaborative work environment. The application-centric approach is not suitable for developing a data schema for a central project/product management system (PMS) because it cannot capture additional information that is required for managing project/product information (which are usually not included in application data structures). On the other hand, the process-centric modeling approach has the strength over the application-centric approach in this regard.

3.3 THE PROCESS-CENTRIC MODELING APPROACH

Process models aim to describe a process in terms of (who-) what-when: e.g., what are the tasks?; what first?; what next?; what are the precedences among activities?; what if?; and sometimes who did what? A *process model* describes how activities within a process are connected, ordered, and structured, and represents a *use case* of information. A process-centric data modeling method is a data modeling method that uses a process model as a means to collect user requirements. Many modern data modeling methods are taking the process-centric approach including the IDEF (NIST 1993) and the UML (Booch, Rumbaugh, and Jacobson 1999), and some ER data modeling15 methods. (See Appendix C and Appendix D for more review on requirements collection methods.)

The advantages of a process-centric and use-case-driven data modeling approaches have been discussed by many studies (Augenbroe 2002; Elmasri and Navathe 2000, 2004; Garg and Jazayeri 1996; Rosenberg and Scott 1999, 1999). Some of them are as follow:

- It represents complex and specific user requirements in a visible and formal description.
- It provides a means to formally review, validate, and improve the requirements.
- It clearly defines the scope of a product data model.
- These capabilities are crucial especially for a large-scale development project.
- The captured requirements can be reused in the update or in similar projects.

[15] In ER data modeling, Data flow Diagrams (DFDs) are often employed rather than a process model. Strictly speaking the DFD method is not a process modeling method because it represents data flow between systems, not between activities.

In addition, if a product data model can be derived directly from collected process information, theoretically the completeness of a product model can be guaranteed. The next section formally defines the relationship between a process model and a product model.

3.4 THE COMPLETENESS OF A PRODUCT MODEL

The basic process-modeling elements include states, activities (tasks or functions), and flows (relations or transitions).

- An activity (A) is a logical step within a process. An activity processes information.
- A state (S) is a mode of a project. The state of a project or information processing is changed by a set of activities (A). A project cannot autonomously change its state.

  $\{A_0, A_1, A_2 \ldots\}(S_i) \rightarrow S_{i+1}$

  where Si is the current state of a project or information processing and S_{i+1} is the next state
- Flows define relations (e.g., transitions) between activities.

The relation between a process model and a product model can be formally defined. All the activities in a process require input information to perform their tasks and yield output information. From a product-modeling point of view, an activity of a process can be defined as *an act of processing information items* (Eastman 1996). An *information item* is a minimum expression of product information. An activity can be formally defined as follows:

Def. 1: $A \equiv \{(i, f) \mid i \in I \wedge f \in F \wedge \exists J(J \subseteq I \wedge J = f(i))\}$

where A is an activity, I is a set of information of a Universe of Discourse (UoD), J is a subset of I, and F is a set of non-decomposable functions or acts of processing information. F produces a new set of information J and receives, generates, updates, deletes, or distributes an information item.

Def. 1a: F = {receive, generate, update, delete, distribute}

Similarly, in this perspective, a *process* is a *set of activities*, *states*, and their *relations*. A relation (i.e., flow) can only connect either an activity and another activity, or an activity and a state at a time.

Def. 2: $P \equiv \{(a, s, r) \mid a \in A \wedge s \in S \wedge r \in R \wedge \exists b \exists t (b \in A \wedge t \in R \wedge (r(a, b) \vee t(a, s)))\}$

where P is a process, R is a set of relations (or flows) between an activity and an activity or between an activity and an activity, A is a set of activity, and S is a set of states

By replacing activities in Def. 2 with sets of information in Def. 1, a process can be characterized by the collection of information processed by its activities.

$P \equiv \{((i, f), s, r) \mid i \in I \wedge f \in F \wedge s \in S \wedge r \in R\}$

A product data model is *a set of information items and their relations*. Note that information items of a product model have different relations (or a structure) from those of a process model. However, if they are describing the same UoD, then the collection of information items should be the same.

Def. 3: $D \equiv \{(i, q) \mid i \in I \wedge j \in I \wedge q \in Q \wedge \exists j (q(i, j))\}$

where D is a product data model, I is a set of information in a Universe of Discourse (UoD), Q is a set of relations between information items in a product model.

If the UoD includes multiple processes, information items in a product model will be equal to the union of every information item in each process.

$I_d \equiv \{i \mid i \in P_0 \vee i \in P_1 \vee \ldots i \in P_n\}$

where I_d: = a set of information in a product model D, P_n is a process

By restructuring (or normalizing) the information collected from each process of the UoD, theoretically a product model can be derived. When one can capture all the activities within a process and information items processed by each activity, a product model derived from the collected information can be said to be *complete.* Thus, if a certain set of information is not included in a product model, it is either because the process model is not properly defined or because the information required by each activity has not been properly specified.

3.5 THE ARCHITECTURE OF GTPPM

This study takes the process-centric product modeling approach because it has many advantages as described earlier and also because it is a standard approach. The new process-centric product modeling method proposed in this thesis is called *Georgia Tech Process to Product Modeling (GTPPM)*. GTPPM consists of two modules: the *Requirements Collection and Modeling* (RCM) module and the *Logical Product Modeling* (LPM) module (Figure 3.1).

RCM is a graphical Requirements-Collection-and-Modeling method for capturing information in the context of its use. A RCM model consists of three parts:

- *process modeling*: Different users (or companies, applications) may use information in different ways. GTPPM (RCM) encourages domain experts to generate a process model based on their current or envisioned work process without compromising other processes.

- *vernacular information items (VII) specification*: Domain experts may specify information used by each activity in their local terms. This task is optional.
- *information constructs (IC) specification*: Information constructs (ICs) are formally defined information items used within a process. Modelers can specify information used by each activity in a formal and standardized (machine-readable) way using ICs. Or they can define VIIs first and then map VIIs to the equivalent ICs. Whatever the case, information items should be defined as ICs in the final collection of information items to support automation of the analysis process.

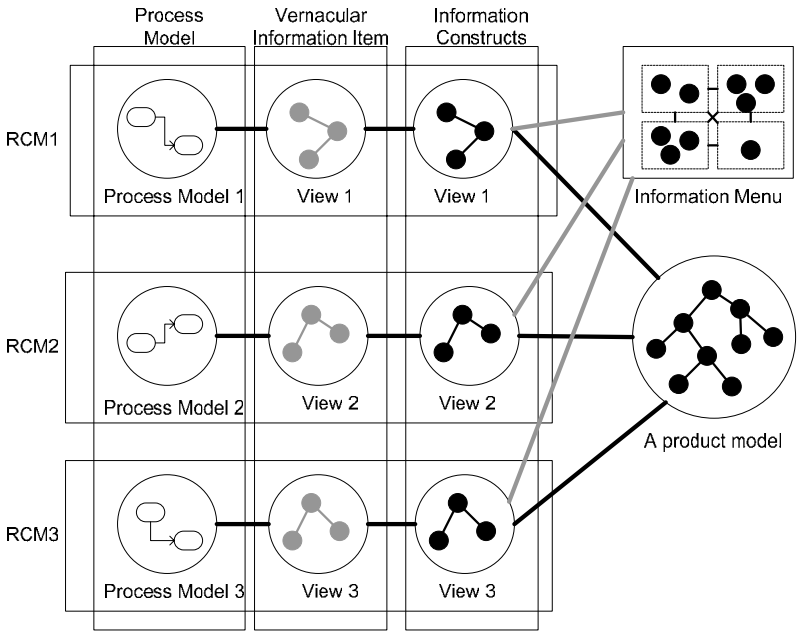


Figure 3.1 The architecture of GTPPM

An *information menu* is a collection of tokens possibly used in a UoD with a classification structure. It restricts the ways in which tokens can be strung together in

constructing information item. A token is a “non-decomposable meaningful lexical element (ISO TC 184/SC 4 1994)”. Tokens in an information menu should be defined following the ‘nym’ principle: ‘no synonyms, no homonyms’ (Schenk and Wilson 1994). A set of rules for developing an information menu has been proposed in Section 4.6.

An information menu and a traditional data dictionary are similar in that both define tokens and their definitions and relations. However, an information menu is different from a data dictionary in several ways. While a traditional data dictionary is a collection of definitions of an existing data model, an information menu is not. An information menu carries only *tokens* and all the *logically possible* relations between them where as traditional data dictionaries carry details of *entities in a final data model* and, sometimes, *fixed* relations between them. For example, a token “door” can be defined as an attribute as well as an entity in an information menu as far as it means the same thing. Also the relationship between tokens is not predefined. The token “door” and another token can be defined as the association relation and also as the specialization relation. Conflicts between the relations and the data types should be resolved in the LPM phase. Another difference between an information menu and a data dictionary is that only a subset of tokens defined in an information menu is included in a product model whereas the set of tokens in a data dictionary is equal to the set of tokens in its data model.

A collection of information constructs or vernacular information items is a *view*, not a *subset* of a final product model (Figure 3.1). The definition of a view is consistent with that of a view in data modeling. A view can be formally defined as a *semantic subset* of its superset similar to the concept of semantic intersection: i.e., a view is a *derivable subset* from its superset.

For example,

Let S be a set of information.

T be a subset of S

V be a view of S

If S ≡ {product_id, product_name, product_volume, product_density},

and T ≡ {product_id, product_volume}

then,

V≡ {product_id, job_name, total_number_of_product, product_weight}

where job_name is product_name,

total_number_of_product is the total count of product instances,

prouct_weight = product_volume × product_density

ICs collected through the RCM phase will be analyzed, integrated, and converted into a product model through the Logical Product Modeling (LPM) phase. LPM is an algorithmic process to derive a product model from collected information constructs. This process is often hidden from users. It's composed of several steps:

- Integration of information constructs (ICs) from several RCM models
- Normalization of collected information constructs into a formal product data model

The next two sections provide detailed descriptions on the RCM and the LPM modules.

CHAPTER 4

REQUIREMENTS COLLECTION AND MODELING (RCM)

4.1 INTRODUCTION

Without clear definition of the required information collected in requirements analysis, a data model cannot be designed to perform its targeted functions. For this reason, in order to facilitate the participation of end-users at an early stage of data model development, techniques such as Joint Application Design (JAD) and Contextual Design (Beyer et al., 1997) have been proposed. Also, several data collecting methods, including a Use Case Driven Approach (Jacobson, Jonsson, and Overgaard 1992), Data flow Diagrams (DFDs), and Upper Case tools are often deployed. However, it is still very difficult to capture a complete set of required information for a model for the following reasons:

- As error-prone human beings, modelers are apt to miss certain requirements.
- Natural language is ambiguous. In a large modeling effort, it is not rare to see one modeler use a term in one way, and another modeler use it in a different way.
- Specific methods to check the consistency and completeness of collected information at an information-level have rarely been introduced. Some methods, including Jacobson's Robustness Analysis (Rosenberg & Scott, 1999), include consistency checking of a model, but they are mostly based on the logic and syntax of diagrams – e.g., a certain shape can be connected

to a shape, but not to the others - rather than on the captured information itself.

Methods that can improve the quality of information generated in the requirements stage can result in higher quality software development. The author proposes a new Requirements Collection and Modeling (RCM) method. The RCM aims to achieve the following goals:

- to model the functional and procedural requirements of a domain for enterprise reengineering and software engineering, using process modeling,
- to systematically collect the rich set of information required for a product model in the context of its use-case scenarios, i.e., a process (Eastman, Lee, and Sacks 2002). The rich set of information should help product-modelers gain in-depth understanding of an industry by:
    - providing accurate definitions of terms
    - providing a complete set of information required for product modeling. By the completeness of a product model, we mean full support and coverage of the Universe of Discourse (UoD)
    - making the semantic differences between terms used in different companies explicit
    - identifying groupings of information used
    - exposing differences in the business practices of different companies
    - supplying various information-use scenarios of each company
- to automatically validate the consistency of the information collected

- to capture the heterogeneous processes of multiple companies within an industry domain
- and to generate a standard product model without losing the unique features of each company's process.

4.2 THE GTPPM RCM LANGUAGE

Like any other graphical modeling language, the RCM has *semantics, syntax,* and *shapes (symbols). Process semantics* dictate the 'meaning of process-modeling components' while *process syntax* dictates the 'structure of process-modeling components'. A *shape* is the 'geometric configuration of process modeling concept'. RCM's notation, syntax, and semantics are based on those of current process-modeling-language conventions so that users can minimize their learning curve and errors. They are basically similar to the definition of traditional workflow (ANSI - IEEE standard 5807-1985, ANSI, 1991) and UML Activity Diagrams. However, the RCM has some unique concepts and syntactic rules in order to allow users to *explicitly* (and sometimes *implicitly*) specify information items used in a process.

As defined in Def. 2 of Section 3.4, a process model is composed of activities, states, and relations between them:

$$P \equiv \{(a, s, r) \mid a \in A \wedge s \in S \wedge r \in R \wedge \exists b \exists t (b \in A \wedge t \in R \wedge (r(a, b) \vee t(a, s)))\}$$

RCM has four types of *activities (A),* three types of *flows (R),* and two types of *states (S).* In order to enrich the process semantics, two variations of an activity (i.e., *static information source* and *dynamic information repository*) that represent information storage and two information flow controls (i.e., *decision, continue)* are added. The following

sections describe RCM components, their syntactic rules, and relations with information in detail.

4.3 ACTIVITIES

An *activity* represents a discrete task. In RCM, activities are categorized by two axes. Activities can be distinguished first as *internal activities* or as *external activities*. Internal activities represent activities that are *within* a UoD while external activities represent activities that are *outside of* a UoD. Many requirement engineering methods focus only on internal activities and often ignore external activities. However, in order to check the consistency of information flow between external and internal activities as well as between internal activities, external activities that are interfacing with internal activities and their information items should also be specified. (See Section 4.8 for details on the consistency checking of information flow.) Thus, external activities are explicitly defined separately from internal activities in GTPPM.

In addition to the external and internal concept, activities can be categorized as *high-level activities* or as *detailed activities.* High-level activities are a relative concept to detailed activities. High-level activities are aggregations of other high-level activities and/or of detailed activities. The hierarchical structure of activities provides a context of the overall model and helps modelers to elaborate a process step-by-step from high-level activities to detailed activities without missing any critical aspects of a model. Among high-level activities, the highest activities are called *top-level activities* (Figure 4.1). A top-level model, composed of top-level activities, is similar to a *context diagram* in a DFD (Data flow Diagram) and a *top-level context diagram* in IDEF0. Note that there is no

separate notation for a top-level activity because top-level activities are merely a type of high-level activity and behave in the same way (Figure 4.1).

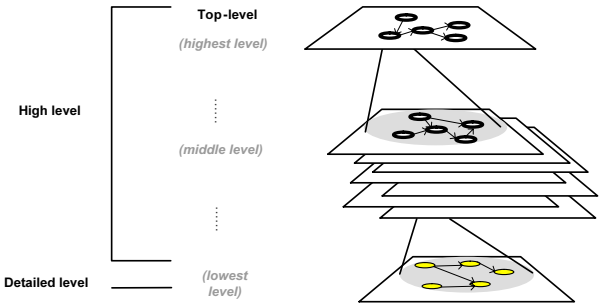


Figure 4.1. The hierarchy of activities

The notation for the combinations of the two distinctions (external/internal and high-level/detailed) is presented in Figure 4.2:

$A \equiv$ {internal highlevel activity, internal detail activity, external highlevel activity, internal detail activity }

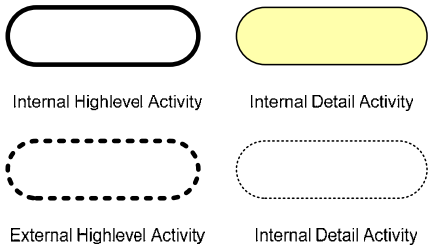


Figure 4.2 Activities

Figure 4.3 illustrates the basic mapping concept between activities and information items. Each activity uses a certain set of information items. Some information items may be used repeatedly, but some may not be used at all. Information items in detailed activities are explicitly defined, but no information items are specified for high-level activities Figure 4.11 for details). This avoids redundancy and potential conflict between the

information recorded in a high-level activity and that detailed in its constituent detailed activities. Instead, the information used in high-level activities can be derived by aggregating the information of their constituent detailed activities.

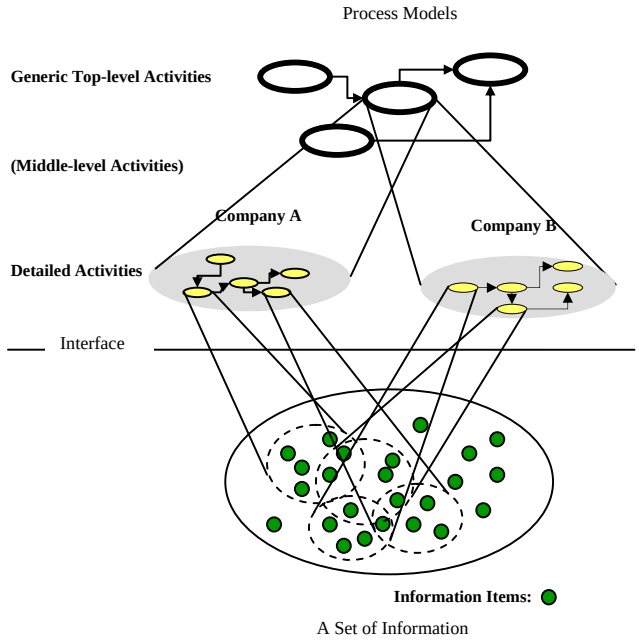


Figure 4.3. A basic mapping concept between process models and an information items

4.4 FLOWS, TRANSITIONS, AND DEPENDENCIES

A *flow* represents the movement of information and objects between activities. In RCM, flows are categorized into *information*, *material*, and *dummy flows* by the information type that they transfer and into *forward* and *feedback flows* by the direction of information flow.

A *material flow* represents a flow of physical objects and information that describes them. An example is a product marked with a bar code carrying encoded data from a plant to storage. Other flows that carry information are *information flows*. Information flows that do not carry *explicitly-specified* information items are called *dummy flows*. Information

flows between external activities or between activities at different levels of detail are dummy flows:

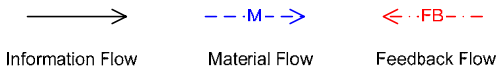


Figure 4.4 Flows

Most modeling methods allow feedback, but they do not generally distinguish feedback from forward flows. However, if workflows are defined at an information level, it is important to distinguish *feedback* from *forward flows* because they imply cyclical repetition of activities.

The following four syntactic rules apply to all types of flows:

Rule 1: A flow can link any shapes except for flows.

Rule 2: A flow must be from one shape to another; it must link exactly *two different* shapes.

Rule 3: A flow must have two distinctive ends to indicate a direction.

Rule 4: Flow arrows can connect activities at any level of detail. However, a flow between activities at different levels is by definition a dummy flow. In order to explicitly describe an information flow between an internal detail activity and any type of high-level activity, a flow must exist between the detailed activity and a constituent detailed activity of the high-level activity in addition to the original dummy flow between activities at two different levels (Figure 4.9).

Feedback flows must conform to the following syntactic rule:

Rule 5: Feedback flows must always participate in the formation of a cycle within a process.

4.5 OTHER PROCESS-MODELING COMPONENTS AND NOTATION

The concepts of the remaining RCM process-modeling components (Figure 8.1) are summarized in the subsections below.

4.5.1 Initial and Final States

Figure 4.5 Initial and final states

Initial and final states represent the starting and ending points of a process. A process embedded in a complex context may have multiple starting and ending conditions, with multiple initial and final states. The state of a process or project is regarded as "in process" if the state of project is omitted between activities. (See Section 3.4 for a formal definition of the relationship between activities and states.)

4.5.2 Static Information Sources

Figure 4.6 Static information source

Static information sources are sets of predefined information of an organization outside of a project. Examples are regional codes, regulations, standards, manuals, etc. A

static information source does not receive, update, delete, or generate information within the context of a project, but only distributes information to descendent activities:

Fs = {distribute}

where *Fs* is a function of Static information sources

cf. Def. 1a: *F* = {receive, generate, update, delete, distribute}

where *F* is a function of Activities.

4.5.3 Dynamic Information Repositories

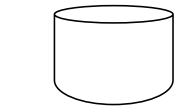


Figure 4.7 Dynamic information source

Dynamic information repositories represent information reservoirs such as project-specific database management system (DBMS) or a schedule board that allow dynamic storage and retrieval of information within a project. Note that only a portion of the information generated and used in a process, is stored in a database and managed. A dynamic information repository only receives, updates, deletes, or distributes information, but does not generate information:

Fd = {receive, update, delete, distribute}

where *Fd* is a set of functions of Dynamic information sources (cf. Def 1a)

4.5.4 Continue

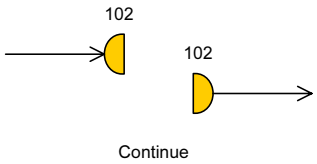


Figure 4.8 A pair of continues

A continue represents the continuity of flow. The main function of a continue shape is to increase readability by interrupting an information flow between two activities, to allow reference across pages or across areas of a model that contains dense graphics. The software aids the user in ensuring that:

- Continue shapes exist in pairs; an “out” and an “in” continue shape. There cannot be multiple flows in or out of a continue shape.
- Each pair of continues must have a unique identifier. And an “in” and “out” pair of continues must use the same identifier.
- Pairs of continue shapes transfer information only between detailed activities.
- When a flow connects an internal detail activity and any type of high-level activity, a continue shape must be placed between two activities to redirect the flow from a dummy flow to an information flow (See Rule 4 for flows and Figure 4.9). In Figure 4.9, an information flow is represented as a thick line to help readers to better understand the diagram

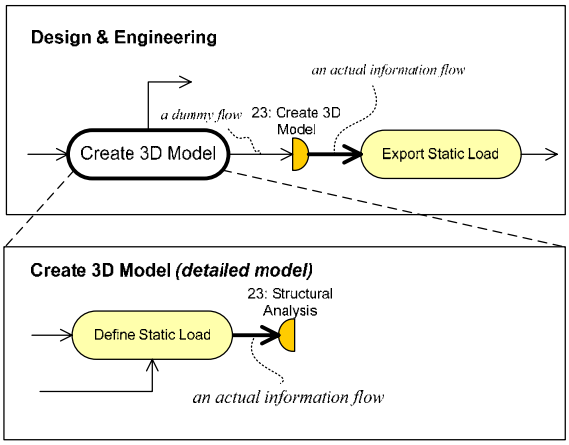


Figure 4.9. A continue shape and a dummy flow between activities at different levels

4.5.5 Decision

A *decision* (control) defines a condition (C) of flows (R: relations) between activities and/or states. Semantically, decisions represent an (exclusive) OR-transition and support what-if scenarios (e.g., “if approved” or “if x > 1”). An OR-transition in RCM includes a decision component, which represents the conditions of the transition.

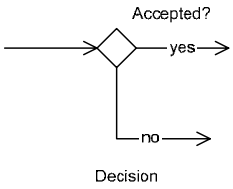


Figure 4.10 Decision

4.5.6 The Process Components and Their Attributes

Each process-modeling component carries certain information. The process components and their attributes are illustrated in Figure 4.11 on the next page using EXPRESS-G. Note that only detail activities, information repositories, and information flows explicitly carry product information. Examples of RCM models are presented in Section 5 Implementation and Examples.

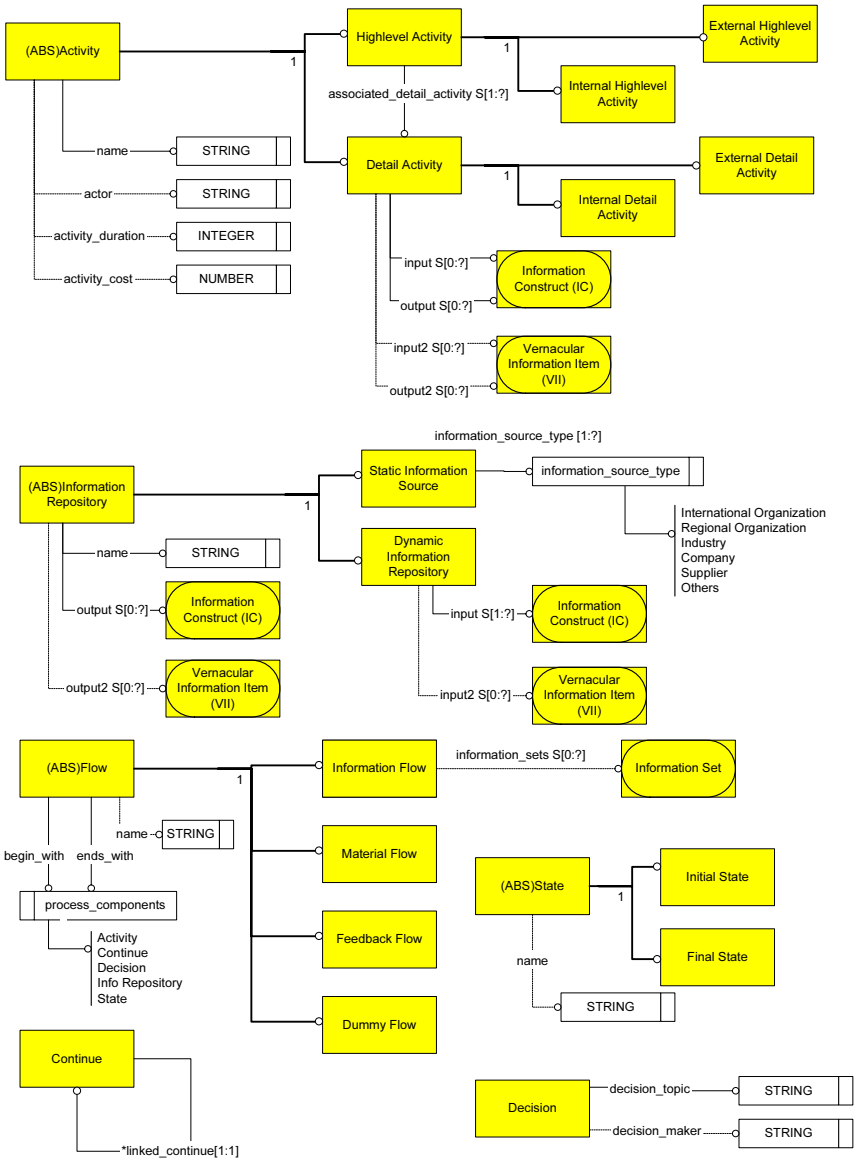


Figure 4.11 Process-modeling components of RCM and their attributes[16]

4.6 A GRAMMAR FOR PRODUCT INFORMATION

The ultimate goal of RCM is to capture "information" requirements for product modeling through process modeling. Product information is the information generated, used, and maintained in the processes of design, engineering, manufacturing, delivery, and

[16] Refer to the GT PPM (Lee, Sacks, and Eastman 2002b) for details. The same component names (without underbars or abbreviation) as those in the texts have been used to help readers to better map them)

maintenance. Examples of product information in the building industry are building type, building identifier, owner first name, and so on. When product information is formally structured, the structured schema of product information is called a product data model (or a product model). (Note that we use two terms a product data model and a product model interchangeably in this paper.) A product model consists of attributes with specialized meaning, special entities and features with technical functions, and aggregations across specialized classes.

This section describes a method to allow domain experts to capture and specify *product information* in a consistent and analyzable format. We call the proposed method the *Product Information Specification (PIS)* method or mechanism. A long-term goal is to (semi-)automatically derive a data model out of the product information specified by domain experts, who know the domain best. However, product information is difficult to capture because of the following reasons:

a) *Tacitness*: Product information is tacit. Even domain experts, who use product information everyday, cannot easily articulate product information required without a specific context.

b) *Enormousness*: Product information has an enormous volume. It not only includes direct geometric and material descriptions, but also all kinds of other information such as that on their design, engineering, manufacturing, and management processes. (For example, the CIMsteel product data model used in the structural steel industry has over 731 entity types covering the design, analysis, shop detailing and fabrication of steel structures for buildings (AISC

2002)). Such a huge amount of product information is very difficult not only to capture, but also to depict in an unambiguous, consistent, and analyzable form.

c) *Informality*: Information can be managed and learned with great ease and efficiency when it is well structured. However, it is not easy to categorize information in an easily recognizable and universally applicable structure when it relies on a grammar of a natural language.

d) *Ambiguities*: When information items are described in a natural language, the collected information will yield *lexical* and *structural ambiguities*. Examples of the lexical and structural ambiguities in product information are:

- *Lexical ambiguity:* Even within an industry that produces the same products, different terms are often used by different people to refer to the same concept or object. For example, in the precast concrete industry, 'control number' is used differently in different companies. In some, it refers to a 'product number', 'production serial number', 'serial number' and so on, which is assigned to a piece after it is fabricated. In others, it is used quite differently, as an 'assembly location number' or 'erection control number', which is used to schedule detailing, production and erection sequences. The lexical ambiguity is also called the *'nym' problems* (i.e., homonyms and synonyms) (Schenk and Wilson 1994).
- *Structural ambiguity:* Often product information is not a single word, but a combination of several words like a *phrase* in natural language. Information items can be constructed in various ways. However, often the richer the expressions are, the subtler the differences between the expressions.

Sometimes subtle differences in the order of terms can make a significant semantic difference. For example, '*concrete finish*' signifies 'finishing applied to a concrete surface' while '*finish concrete*' signifies 'a special concrete used as a type of finish for a piece.'

In order to overcome these difficulties, many formal knowledge specification methods and languages have been proposed and developed, especially in knowledge representation (KR) and data modeling. Examples of the formal specification languages for knowledge-based systems (KBS) include DESIRE, FORKADS, KbsSF, (ML)2, MODEL/KADS, MoMo, OMOS, QUL, and KARL. Some formal approaches for data modeling are the Relational Model (Codd 1970), the Entity-Relationship Model (Chen 1976), the Functional Data Model DAPLEX (Shipman 1981), the SDM (Hammer and McLeod 1981), the Object-Oriented Model (Banerjee et al. 1987) and other semantic models. These methods gave birth to several (standard) data modeling languages such as SQL (ISO JTC 1/SC 32 2003), IDEF1x (NIST 1993), XSD/XML (Berners-Lee 1994; Cover 1999), and a standard product data modeling language EXPRESS (ISO TC 184/SC 4 1994; Schenk and Wilson 1994). The data modeling languages listed above have been refined over decades and have their strong adherents. Nevertheless, we found that existing former data modeling and KR methods are not suitable for our purpose (i.e., specifying product information in a simple, yet consistent and analyzable form) because of the following reasons:

- Specialized product information is often carried as implicit knowledge in natural language through everyday conversation by domain experts. We believe that domain experts are the best persons to describe product

information required for their tasks. But many formal modeling languages are not generally accessible by domain experts. Some modeling languages are even close to mathematical descriptions.

- Modeling languages such as XSD or XML may be simple enough to be used by domain experts even in the very early data modeling phase: i.e., the requirements collection phase. However, they still do not provide a mechanism to maintain the consistency (i.e., the lexical clarity) of an enormous amount of terms used in a UoD. (The limitation of XSD and XML in expressing the semantics of the specialization (inheritance) relation is another issue here.)

Note that the PIS method, we are proposing in this paper, is not to develop a generic structure of product models such as ISO STEP Part 41 (ISO TC 184/SC 4 2000) and the Generic Core Representation of product information (Szykman et al. 2001). Also its goal is not to define a data dictionary for product information such as the STEP Library (Renssen 1997) or to propose another data modeling language, which can replace XSD or SQL. The proposed protocol is independent of data modeling languages and can be implemented in XSD (XML), SQL, EXPRESS, or any other data modeling languages later albeit we chose EXPRESS as a main target because EXPRESS is an international standard product data modeling language by the ISO – International Organization for Standardization (ISO TC 184/SC 4 1994). Rather, it aims to develop a high-level product information categorization and a grammar that can allow domain experts to easily, efficiently, and clearly specify product information in an analyzable form so that the

collected information can be analyzed and transformed into a product data model in the later stage. The criteria for the PIS method can be summarized as follows:

1) *consistency between terms*: There should not be the ‘nym (homonyms and synonyms)’ problems and ambiguities in the definitions of terms.
2) *generativity* & *extensibility*: The list of product information should be extensible and editable, and not fixed. Domain experts should be able to generate and add new information constructs as many as possible.
3) *analyzability*: Information constructs built from an information menu should be analyzable and transformable to a form of a product model.
4) *accessibility*: An information menu should be structured in a way that domain experts (non-data-modeling experts) can easily navigate and maintain a large amount of product information.

The following sections describe the concept of the PIS mechanism in more detail. And they also discuss how to construct a system of rules that both analyze and generate structured product information.

Product information is basically a concatenation of *tokens* (or words). A *token* is a “non-decomposable meaningful lexical element (ISO TC 184/SC 4 1994)” of a UoD. Examples of tokens are ‘width’, ‘job’, ‘height’, and ‘color’. A token *per se* (e.g., ‘type’) has a certain meaning, but often is insufficient to represent *product information*. On the other hand, if several tokens are concatenated in a logical way, the chain of tokens can represent meaningful product information (e.g., ‘finish-material-type’, ‘engine-type’). This paper explores and defines grammatical rules for specifying product information by concatenating tokens in a consistent and analyzable form, similar to grammatical rules for

generating syntactically and, sometimes, semantically meaningful sentences (or phrases) in a natural language. The following sections describe the concept of the PIS mechanism in more detail. And they discuss a system of rules that both analyzes and generates structured product information.

This study takes a linguistic approach in defining the structure and the syntactic rules for defining product information. A linguistic approach (i.e., the *context-free grammar* (CFG)) is taken because (1) a data model is essentially a representation of the universe of discourse (UoD) based on a language; and (2) even 40 years after the CFG was first introduced by Chomsky, it is still an effective and efficient means to analyze and define grammatical rules for generating meaningful expressions. The proposed system will be a duplex <B, R> consisting of a set B of basic elements and a set R of *context-free rewrite rules* each of which defines a minimal hierarchical structure, called a *local tree* (Chomsky 1965, Ch 1-2; Smith and Wilson 1979). Appendix E provides a brief summary of notation of a context free grammar (CFG). Some notational rules are revised or added to suit the characteristics of product information and the purpose of this study.

4.6.1 Product Information Structure and Grammar

The RCM PIS method categorizes product information at three levels, namely *tokens, information items* and *information sets* and provides a grammar for defining product information.

As stated earlier, a *token* is a “non-decomposable meaningful lexical element (ISO TC 184/SC 4 1994)” of a UoD. Examples of tokens are ‘width’, ‘job’, ‘height’, and ‘color’. A token *per se* (e.g., ‘type’) has a certain meaning, but often is insufficient to represent *product information*. On the other hand, if several tokens are concatenated in a logical way,

the chain of tokens can represent meaningful product information (e.g., ‘finish-material-type’, ‘engine-type’). We call the concatenation of tokens an *information construct* (IC). The definition and the structure of tokens are recorded in an *information menu*. An *information menu* is a collection of tokens that forms a minimum expression (or phrase) of product information. The differences between an information menu and a traditional data dictionary are discussed in Section 3.5. Figure 4.12 illustrates how product information can be defined using tokens in an information menu. Let us assume an information item “an identifier of a beam, which is a kind of (precast concrete) piece” is required by an activity “Prepare Initial Quotation”. It can be defined as `piece*beam{id}` using three tokens `piece`, `beam`, and `id` in an information menu.

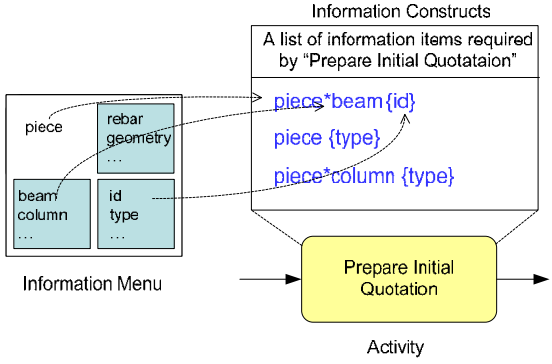


Figure 4.12 An information menu and information constructs

As briefly described earlier, it is assumed that RCM will include two groups of experts, which have expertise in different domains. The two groups are *domain experts* (representatives of an industry of a company) and *modeling experts* (or mediators; process and product modeling experts). Information is classified in a way that can help each group to contribute what it knows best. The product information can be expressed in two ways: either as *vernacular information items* (VIIs) or as *information constructs* (ICs). Domain

experts, who may not be familiar with the structure of an information menu and ICs, can define product information as vernacular information items (VIIs) as far as they provide a definition of the VIIs in their own data dictionary. Later, modeling experts can map ICs and VIIs based on the definitions of VIIs specified by domain experts. Table 4.1 lists examples of mapping between VIIs and ICs. Company *A* may call an identifier of a (precast concrete) piece "Piece Mark." Company *B* may call the same thing "Mark Number." The VIIs are synonyms and can be mapped to an IC "`PIECE{id}`", which is a concatenation of two tokens, i.e., `piece` and `id`.

Table 4.1. Mapping between vernacular information items and information constructs

Company *A* VIIs	**Company *B* VIIs**	**ICs**
Site name	Construction site name	SITE{name}
Site address	Construction site location	SITE{address}
Estimated weight	Load	PIECE+LOADS{weight, unit}
Piece mark	Mark number	PIECE{id}
Serial number	Control number	PIECE{control_id}

The specified information items (both VIIs and ICs) can be grouped as an *information set*. An *information set* is a user-defined grouping of information items that flow from one activity to another. Examples are forms, work order, bills of materials, and specific drawings. Information sets play the following roles in RCM:

1) In everyday life, domain experts do not deal with their work at an information-item level but at an information-set level (e.g., forms, work orders). Grouping information items in sets provides a cognitive bridge between what they actually deal with (information sets) and what they unconsciously process (information items).

2) Information sets can be considered as milestones of information production in a process. An information set implies that its subsumed information items are required in order to proceed to the next activities.

Tokens are further categorized into *types* and *entities*. A type and an entity in this paper are the same as those defined in EXPRESS. A *type* is a "representation of a domain of valid values (International Organization for Standardization 1994)" and an *entity* is a "*type* which represents a collection of conceptual or real-world physical objects which have common properties (International Organization for Standardization 1994)." A set of entities that describes the main physical objects of a domain forms the backbone of an information grammar. For example, *structures, assemblies, pieces, reinforcement* and *embeds* are the *main products* or *parts* of the Precast Concrete Industry.

The structures and relations of different types of tokens are defined in an *information menu*. Modelers are restricted to select information from the limited number of *possible tokens that can be linked* in an *information menu* based on *context-free rewrite rules* (Chomsky 1965; Jurafsky and Martin 2000) defined for product information.

The approach in this study defines *tokens* used in a universe of discourse (UoD) by *four general abstraction mechanisms* of knowledge representation (KR): i.e., *classification & instantiation, aggregation & decomposition, generalization & specialization, and association* (Eastman 1999; Elmasri and Navathe 2000; Smith and Smith 1977; Smith and Smith 1997).

Some early papers (Codd 1979; Smith and Smith 1977) categorize both *instantiation* and *subtype* as a form of *specialization*, but this paper uses the term *specialization* only to represent the *subtype-supertype* relationship. For example, 'bolt' and

‘weld’ are specialized types of ‘fastener.’ *Generalization* is the inverse of specialization. *Instantiation* represents the *is-an-instance-of* relationship. If twelve ‘C8’ chairs are placed in an office, each individual chair is an instance of the chair type ‘C8.’ Note that an instance of a class (i.e., the twelve ‘C8’ chairs) can be either a class or a value of an attribute depending on a modeler’s intention. *Classification* is the inverse of instantiation. *Decomposition* represents the *is-a-part-of* relationship. The inverse is *aggregation* and represents the *has* relationship. A ‘table’ *has* four ‘legs’ and a ‘tabletop.’ *Association* represents other attributive and referential properties. For example, ‘color’ and ‘width’ can be properties of a ‘tabletop.’ The difference between aggregation and association is that when an instance of a higher-level entity in an aggregation relationship is deleted, in some cases its lower-level instances are also deleted: i.e., an aggregation relationship often represents a *semantic dependency* between two entities. For example, if an instance of a ‘table’ is deleted, the instances of its ‘legs’ and its ‘tabletop’ should also be deleted. Entities in an association relationship, on the other hand, do not need to be deleted even when their associated entities are deleted. *Identification*, “the abstraction process to define whereby classes and objects are made uniquely identifiable by means of some identifier” (Elmasri and Navathe 2004) can also be added to these four abstraction concepts.

Currently EXPRESS is a standard language for specifying a *product data model* (ISO TC 184/SC 4 1994). Since the eventual goal of GTPPM is to develop a product model in EXPRESS, the RCM PIS method should comply with the structure of EXPRESS. EXPRESS supports the three abstraction mechanisms (i.e., *instantiation, specialization, and association*). EXPRESS does not distinguish the aggregation & decomposition relation from the association relation. EXPREES takes an object-oriented approach. Naturally, the

classification & instantiation relation is embedded in EXPRESS. However, in EXPRESS, the term *instantiation* is used generally to represent *data population* similar to the instantiation concept in object-oriented programming language. EXPRESS does not distinguish the instantiation relation between classes from *subtyping*. Both the instantiation relation between classes and subtyping are regarded as a type of *specialization*. The generalization & specialization relationship is defined by the SUBTYPE OF and SUPERTYPE OF constraints in EXPRESS. And the *classes* and their *instances* are defined as the ENTITY and ATTRIBUTE constructs and their values. The association relation includes all other relations between ENTITIES and ATTTRIBUTES. Although EXPRESS does not distinguish the decomposition relation from the association relation, the proposed method classifies entities in the decomposition relation differently from those in the association relation.

EXPRESS has four *existence constraints*: BAG, LIST, SET, and ARRAY and the *cardinality ratio* (or *arity*): e.g., LIST [0:?] OF and SET [1:?] OF. These can be imposed between ENTITIES with the association relationship. In data modeling, the existence constraints (esp. cardinality ratio) and other types of constraints (e.g., RULES) are often defined in the late phase of logical data modeling. This PIS method focuses on the early requirement collection phase of data modeling and, is therefore relatively unconcerned with detailed level *constraints* (e.g., the existence constraints) between information items.

4.6.2 Categorization of product information

We first categorized product information in a fashion similar to categorization of *parts of speech* such as nouns, objects, and adjectives in natural language before establishing rules for specifying consistent and analyzable product information.

By the definition of product model, constituents of any product model well accepted today (e.g., ISO STEP (ISO TC 184/SC 4 2004) and IFC (IAI 2003)) can be categorized into *information that directly represent products* and *information that qualifies products*. We call the former *product entities (P)* and the latter *modifiers (M)*. Based on this distinction, tokens, which compose a product information item, are first categorized into two major *abstrac*t constituents: *product entities* (P) and *modifiers* (M). The definition of *entity* in 'product entities' is compliant to that of ISO 10303 (International Organization for Standardization 1994): an *entity* is a "*type* which represents a collection of conceptual or real-world physical objects which have common properties" and a *type* is a "representation of a domain of valid values." An entity without properties is called an *empty entity*. The definition of *empty* is identical to that in mathematical set theory: i.e., a set without an element. An entity cannot be empty and must have a property:

Rule 1: Unless an entity inherits properties from its higher-level entities, an entity must not be *empty*.

Product entities (P) literally represent entities describing the products of an industry. A modifier (M) is either an entity or an attribute that "qualifies" product entities (P) or other entities. The "qualification" relation between a Product entity (P) and a modifier (M) is often represented as the association relation, but sometimes can be represented as the specialization relation. (An example is provided in the next section.) An attribute is a trait or property of an entity. Modifiers (M) describe the design, engineering, manufacturing, and management information of products. Modifiers are subcategorized into Modifier Entities (ME, an entity-type modifier) and Modifier Attributes (MA, an attribute-type

modifier) by their type. An example of a product entity and a modifier is 'CAR' (a product entity) and 'DESIGNER' (a modifier).

The definition of a product entity is relative. It depends on the universe of discourse. 'CAR' is a product of the automobile industry, but 'BUILDING' is not. 'BUILDING' is a product of the building industry, but 'CAR' is not. 'DESIGNER' is not a product of the automobile industry, but it provides additional information on a product 'CAR'. Whether 'DESIGNER' is defined as an attribute of 'CAR' or not, 'DESIGNER' still semantically qualifies a product entity 'CAR' and is, therefore, a modifier of 'CAR'.

The product entities (P) and modifiers (M) are further subcategorized by the three major abstraction concepts (Eastman 1999; Elmasri and Navathe 2004): i.e., a) generalization & specialization; b) classification & instantiation; c) aggregation & decomposition; and d) association. Applying these abstraction concepts, product entities (P) are further subcategorized into decomposed products (DP) and specialized products (SP). Decomposed products (DP) represent products in the aggregation relationship. Many researchers (Codd 1979; Smith and Smith 1977) and modeling language including EXPRESS, as described earlier, do not distinguish the specialization (supertype - subtype) relation from the instantiation relationship at a conceptual. Specialized products (SP) represent products in both the specialization relation and the instantiation relationship.

By the same logic, modifier entities (ME) are further subcategorized into specialized modifier entities (SME). Figure 4.13 illustrates a hierarchical structure of PIS information structure in EXPRESS-G. Note that this structure is different from a constituent structure tree and does not imply any syntactic rules.

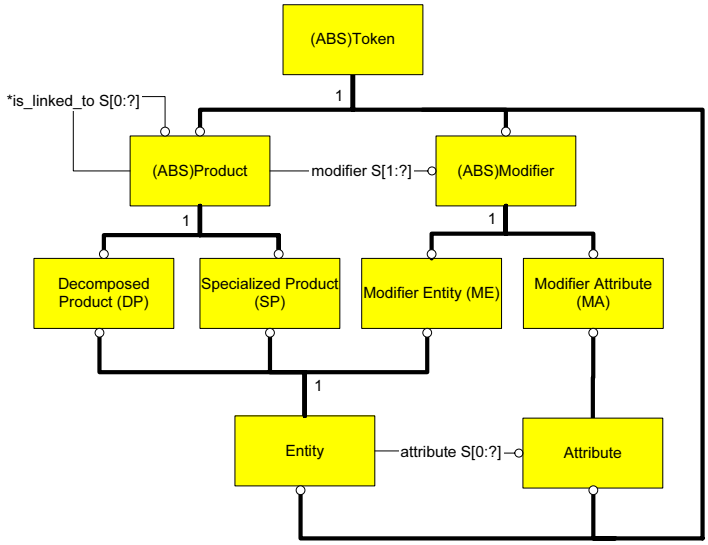


Figure 4.13. A hierarchical structure of RCM product information in EXPRESS-G

4.6.3 Syntactic rules for product information

This section describes syntactic rules for constructing product information by combining product-information constituents categorized in the previous section. We call a product information item composed of several tokens an *information constructs (IC)*. Each information construct (IC) corresponds only to one product information item. Figure 4.14 illustrates two simple ways of composing information constructs.

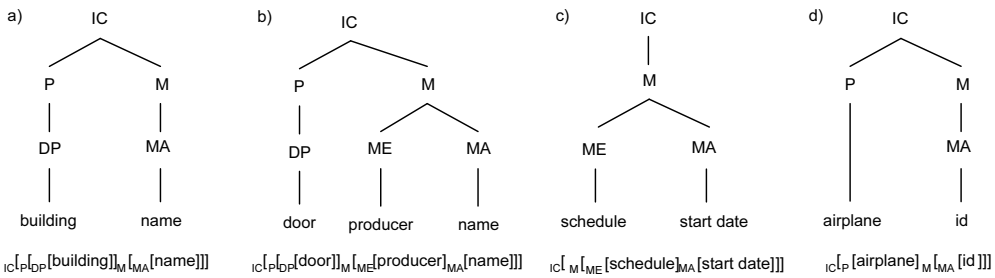


Figure 4.14. The basic constituent structures of an information construct

Rule 2: An information construct (IC) ends with a modifier attribute (MA) (because there cannot be an empty entity).

Rule 3: An information construct (IC) must not end with a modifier entity (ME).

Rule 4: Product entities (P) work as main access points to other information types. If any type of product entity (P, SP, or DP) exists in an information construct, the information construct (IC) always begins with a product entity. If not, the information construct (IC) begins with a modifier entity (ME).

The rules for Figure 4.14 can be summarized by the CFG notation as follows:

IC → P – M | M

P → DP

M → MA | ME – MA

(NB: A vertical bar | denotes "OR".)

As stated earlier in Rule 4, the PIS method defines product information types (i.e., P, DP, and SP) as a kind of *index* for modelers to access other types of product information. It is because product information is the focus of product modeling (thus, any product model includes product information) and also because domain experts are generally very familiar with a hierarchical structure of their product information. In other words, even if non-product information types were used as an access point to product information, it would not make much difference in terms of representing a structure of an information construct. For example, an information item "the delivery date for a column, which is a kind of product" can be represented in two ways: (a) one starting from product information and (b) the other starting from the delivery schedule.

(a) `product*column+delivery_schedule{delivery_date}`

(b) `delivery_schedule{delivery_date}+product*column`

They may require slightly different syntactic rules. But when they are represented as information constructs, they eventually represent the equivalent structure (Figure 4.15 (a) and (b)).

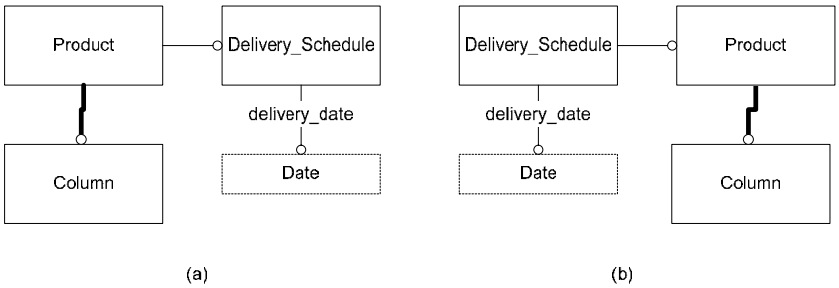


Figure 4.15 Product information as an access point to other information types

Yet the PIS method defines product types (i.e., P, DP, and SP) as a kind of index for modelers to access other types of product information for two reasons. It is because a product and its components are the main focus of product modeling and also because domain experts are generally very familiar with a hierarchical structure of their product information.

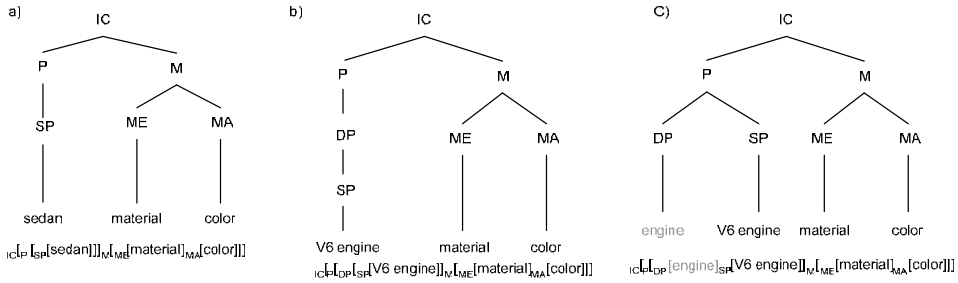


Figure 4.16. Abbreviation of specialized products

Figure 4.16 illustrates *abbreviation rules for specialized products*. The purpose of abbreviation rules is to remove redundant expressions in an information construct so that

the information construct can be expressed in a succinct manner that users of the information can comprehend quickly and easily.

Rule 5: (*abbreviation rules for specialized products*) A specialized product (SP) inherits all the properties of its higher-level entities (i.e., supertypes). Therefore, semantically and logically, a specialized product (SP) alone can represent a product. As exemplified earlier, 'car-sedan' means the same thing as 'sedan'. Thus, we can abbreviate 'car-sedan' to 'sedan' without diluting its meaning.

The applied abbreviation rules can be analyzed as follows: 'car' can be categorized as a main product (P) and 'sedan' can be categorized as a specialized product (SP). A specialized product (SP) can be regarded as a replacement of a decomposed product (DP). Figure 4.16 a) illustrates the first case. The rule applied here can be defined as follows:

Rule 5.1:

P → SP, *iff* SP is a specialized product of P.

The same logic can be applied to the abbreviation of a chain of decomposed and specialized products in Figure 4.16 b) and c). 'engine (DP)' *is a part of* 'car (P)'. 'V6 engine (SP)' *is a type of* 'engine (DP)'. 'engine-V6 engine' can be abbreviated to 'V6 engine' without losing its meaning. The rules applied here can be analyzed in two ways. First the abbreviation phenomena can be analyzed as the replacement of DP by SP as illustrated in Figure 4.16 b). The rules can be described as follows:

Rule 5.2:

P → DP

DP → SP, *iff* SP is a specialized product of DP.

Alternatively, the abbreviation phenomena can be analyzed as the replacement of DP by NULL as illustrated in Figure 4.16 b). The rules can be described as follows:

Rule 5.3:

P → DP – SP

DP → NULL

SP → V6 engine

Both approaches are logically valid and yield the same result: i.e., P → SP. However, the second approach leaves the possibility of having a non-abbreviated form of the information item (e.g., engine-V6engine) while the first approach does not allow any non-abbreviated form of the information item. Thus, the second approach has been taken.

By the same token, a specialized product (SP′) of a certain specialized product (SP) can replace its antecedent specialized product (SP) (i.e., supertype). Applying these rules, a series of specializations can be replaced by the last specialization.

Rule 5.4:

SP → SP – SP′, *iff* SP′ is a specialized product of SP

SP → NULL, *iff* SP is followed by SP′.

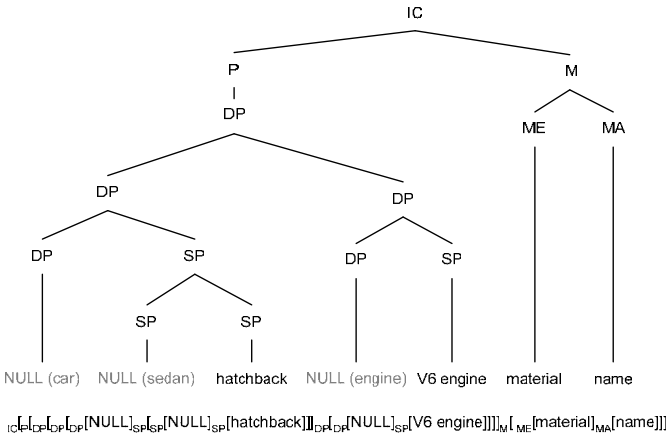


Figure 4.17. Concatenation of specialized products (SP) from different decomposed products (DP)

If the order of tokens is changed, the meaning of an information construct differs. An example is ‘hatchback – V6 engine – material – name’ and ‘V6 engine – hatchback – material – name’. The former depicts ‘the material name of a V6 engine in a hatchback-style car’ while the latter would depict ‘the material name of a hatchback-style V6 engine’ if there were such a thing. Therefore, we set up a rule that says:

Rule 6: In a concatenation of DP – DP′, the DP′ should always be a component of DP.

The rule can be formalized as follows:

DP → DP – DP′, *iff* DP′ is a component of a decomposed product DP

Similarly,

Rule 7.1:

P → DP – DP′, *iff* DP′ is a component of a decomposed product DP

By these rules, ‘hatchback – V6 engine’ should always be interpreted as “a V6 engine in/of a hatchback-style car”.

Figure 4.18 shows an example of *abbreviation rules for decomposed entities*:

Rule 7: (abbreviation rules for decomposed entities) When a series of decomposed products (DP) are concatenated, the last decomposed product represents the whole concatenation.

A formal descriptions of the additional rule is:

DP → NULL, when DP is followed by its decomposed product, DP′

In Figure 4.18, since it is apparent that ‘structure’ belongs to a ‘site’ and a ‘project,’ ‘project – site – structure’ can be replaced by ‘structure.’

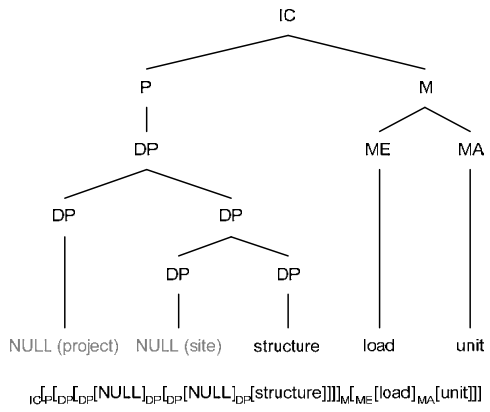


Figure 4.18 Abbreviation of decomposed products (DP)

The same logic for abbreviation rules can be applied to specialized modifier entities (SME). Abbreviation rules for specialized modifier entities are:

Rule 7.1:

SME → SME – SME′, *iff* SME′ is a subtype of SME

SME → NULL, *iff* SME is followed by its subtype SME′.

ME → NULL, *iff* ME is followed by its SME.

For example, if we want to describe the ‘date when a beam was cast,’ it can be expressed as:

IC[P[DP[piece]SP[SP[flexural piece]SP[beam]]]M[ME[ME[production]SME[cast]]MA[date]]]

Applying the abbreviation rules, the information construct can be simplified as:

IC[P[DP[NULL]SP[SP[NULL]SP[beam]]]M[ME[ME[NULL]MAE[cast]]MA[date]]]

≡ IC[SP[beam]M[MAE[cast]MA[date]]]

The constituent structure tree of the ‘date when a beam was cast’ is illustrated in Figure 4.19.

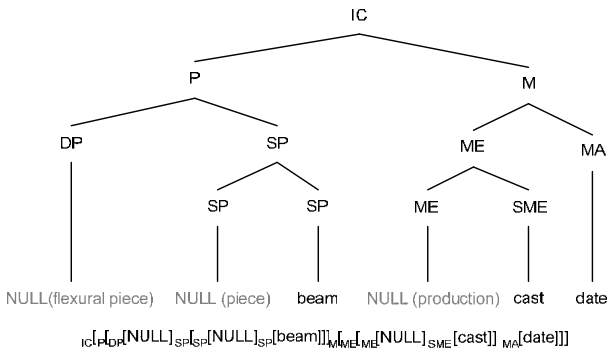


Figure 4.19. Abbreviation of specialized modifier entities (SME)

Not that, in any case, the abbreviation of information constructs is optional, but not mandatory. The main purpose of the abbreviation rules is to recognize semantically equivalent information constructs. Therefore, the use of abbreviation should be minimized. Otherwise, it can yield other ambiguous cases as in natural language.

In GT PPM, the specialization relation has been distinguished from the association relation by using a separate concatenation symbol: An asterisk (*) denotes the specialization relation; A plus sign (+) denotes the association relation. The decomposition relation has not been distinguished from the association relation because the target language EXPRESS does not distinguish between them. (See Section 2 for details.) However, this is an implementation-level decision; if necessary, it is possible to use different concatenation symbols for different abstractions. An example of the ‘date when a beam was cast’ in Figure 4.19 can be represented as:

```
piece*beam+production*cast{date}
≡ piece*beam+cast{date}
≡ beam+cast{date}
```

A full definition of this grammar and its use is being prepared.

4.6.4 Styles of Product Models

Each product model has a *style*, which is also called a *modeling philosophy*, *intention*, or *concept*. Depending on a modeling style, a different *generic structure* of a model (a.k.a. a *core representation*, a *framework*, or a *skeleton* of a model) is created. As stated earlier, the PIS method aims to support any product model defined in EXPRESS. Since the PIS method categorizes product information by generic knowledge representation concepts and by the structure of EXPRESS, the structure of product information defined by the PIS method should be transferable to a product model defined in EXPRESS and also *vice versa*. However, the PIS method itself only defines the rules to structure product information, not what the structure of a final product model should be. The structure of a final product model is defined by how a modeler categories tokens. A structure and a style of product information specified by a modeler through the PIS method will be kept through the GTPPM process and will form the core structure[17] of the final product model. This section shows how various styles of existing product models can be supported by the PIS method in the early requirements collection phase of product modeling. The first example is the IFC 2x2 model. It adopted the top-down modeling approach. As described in Section 2.4, the `IfcRoot` is at the top of the IFC model. `IfcRoot` has three subtypes: i.e., `IfcObject, IfcPropertyDefinition,` and `IfcRelationship`.

```
ENTITY IfcRoot
 ABSTRACT SUPERTYPE OF(ONEOF(IfcObject, IfcPropertyDefinition,IfcRelationship));
 GlobalId   :   IfcGloballyUniqueId;
 OwnerHistory   :   IfcOwnerHistory;
 Name   :   OPTIONAL IfcLabel;
```

[17] The structure of information constructs may not exactly the same as that of the final product model because, if there are conflicting definitions (structures) of product information, those have to be resolved. Also through a normalization process, the structure may vary.

```
 Description   :   OPTIONAL IfcText;
UNIQUE
 UR1   :   GlobalId;
END_ENTITY;
```

And IfcProduct is defined as a subtype of IfcObject.

```
ENTITY IfcObject
 ABSTRACT SUPERTYPE OF(ONEOF(IfcActor, IfcControl, IfcGroup, IfcProcess, IfcProduct,
IfcProject, IfcResource))
SUBTYPE OF (IfcRoot);
 ObjectType   :   OPTIONAL IfcLabel;

INVERSE
 IsDefinedBy   :   SET OF IfcRelDefines FOR RelatedObjects;
HasAssociations   :   SET OF IfcRelAssociates FOR RelatedObjects;
HasAssignments   :   SET OF IfcRelAssigns FOR RelatedObjects;
Decomposes   :   SET [0:1] OF IfcRelDecomposes FOR RelatedObjects;
IsDecomposedBy   :   SET OF IfcRelDecomposes FOR RelatingObject;

WHERE
 WR1   :   SIZEOF(QUERY(temp <* IsDefinedBy | 'IFCKERNEL.IFCRELDEFINESBYTYPE' IN
TYPEOF(temp))) <= 1;

END_ENTITY;
```

In the case of the IFC 2.2x model, IfcRoot, IfcObject, IfcProduct, and other subtypes of IfcProduct can be defined as Product Entities (P). And all other entities and attributes including IfcPropertyDefinition, IfcRelationship, IfcActor, IfcControl, IfcGroup, IfcProcess, IfcProject, and IfcResource can be defined as Modifiers (M).

The IfcRoot and IfcObject entities will be “shared” as supertypes of both Product entities (P) and Modifiers (M). Semantically, IfcRoot and IfcObject are ABSTRACT SUPERTYPEs

of Product entities (P) and Modifiers (M). Technically `IfcRoot` and `IfcObject` will be treated as Product entities (P) and also be an access point to other information types.

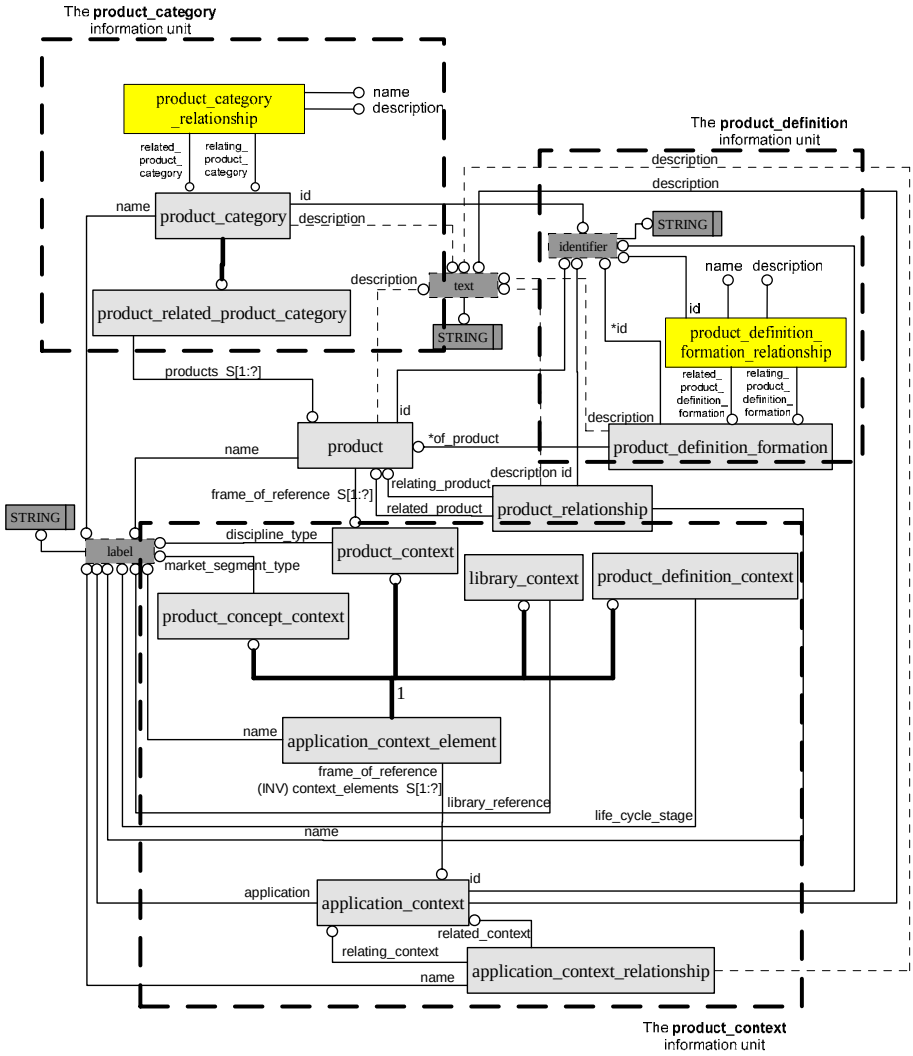


Figure 4.20 A partial EXPRESS-G diagram of ISO STEP Part 41

The second example is ISO STEP Part 41 (ISO TC 184/SC 4 2000). It defines the Generic Product Data Resources (GPDR): i.e., the “information units” for a product model and their interrelated relations. An *information unit* is “a grouping of relating constructs (entity data types, attributes and relationships) that together represent one of the high level

concepts of the STEP data architecture (Fowler 1996)". In Part 41, the product information is represented as the product and product_context entity types at the top level. The product entity type defines a product as being of interest. The product_context is defined from three points of views: 1) the *classification* view: how the product is classified or categorized; 2) the *marketing* view: how the product is presented to the market; and 3) the *technical* view: how the product is defined at a particular life-cycle phase (Fowler 1996).

Figure 4.20 illustrates a partial EXPRESS-G model of ISO STEP Part 41 focusing on the product entity type. The product_category information unit, the product_concept information unit, and the the product_definition information unit in Figure 4.20 respectively represent the classification view, the marketing view, and the technical view.

The mapping between the structure of ISO STEP Part 41 and the PIS method is fairly straightforward. product and its subtypes and subsystems can be categorized as Product entities (P) and the others including product_context, product_category, and product_definition can be categorized as Modifiers (M).

Szykman et al. (Szykman et al. 2001) proposed another generic structure for product information. The proposed data structure is called *the core representation.* The core representation is categorized into DRP_Object and DRP_Relationship at the top level. The DRP_Objects is specialized as Aritifact, Restricted_DRP_Object, Behavior, and Specification (Figure 4.21). And the Restricted_DRP_Object is specialized as Flow, Form, Function, Geometry, and Material by the *function*, *form* (*structure*), and *behavior* concept (Chandrasekaran 1994). In this structure, the DRP_Object and Arifact entities can be categorized Product entities (P) and the others as Modifiers (M). However, the entities can be categorized differently depending on a modeler's intention.

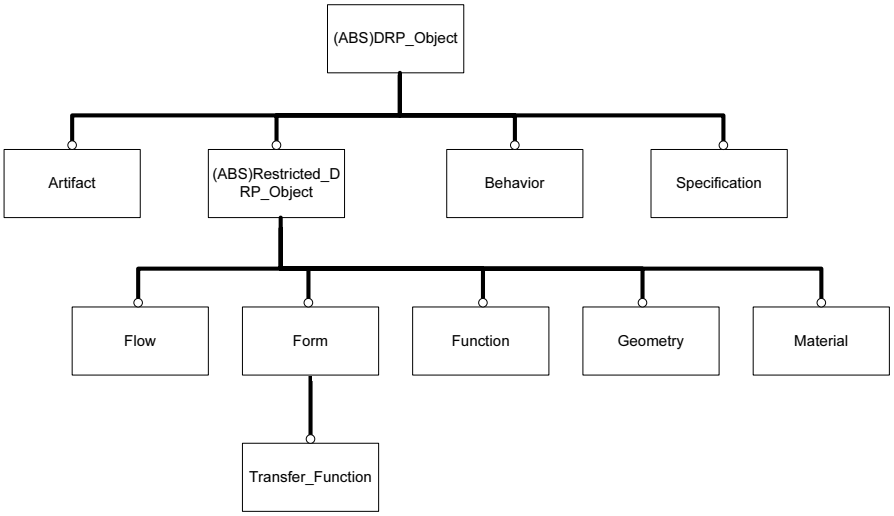


Figure 4.21 The DRP_Object structure of the core representation

Figure 4.22 provides another example of constructing product information created by the author as a proposal for a generic product model structure. The main concept is to structure product information by phase of a product's life-cycle. Figure 4.22 depicts a breakdown structure of main product entities. The vertical axis represents the aggregation relationship and the horizontal axis represents the specialization and instantiation relationship. The specialized products in this paper are structured based on the incremental product design and engineering processes.

As noted earlier, a product (P) can be classified differently depending on its use. Also the depth of layers and the strata can differ depending on the design intention/scope of a data model and the characteristics of products.

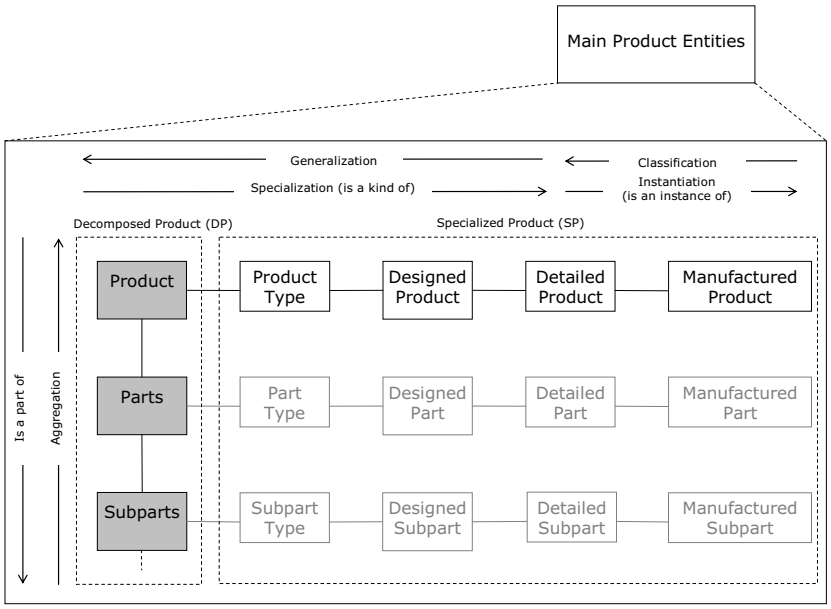


Figure 4.22. An example of constructing product information

4.7 RELATIONS BETWEEN INFORMATION CATEGORIES IN GTPPM

In order to understand the LPM process, readers should understand the relations between GTPPM information types first.

First, a Vernacular Information Item (VII) is semantically equivalent to an Information Construct (IC). This rule is a basis for mapping between a VII and an IC.

Second, an aggregation of information items used in a process is a *view* of a product model. For example, an IC, `PIECE+MATERIAL*CONCRETE{strength}` may look like Table 4.2 in a final product model in EXPRESS:

Third, by definition, entities and attributes in a product model are a *subset* of tokens defined in an information menu. An information menu defines *tokens*, which can be later translated into entities and attributes of a final product model, and the *semantic relations* between them. Tokens, which are not defined in an information menu won't appear as an entity or as an attribute in a product model unless a product modeler intentionally add new

entities or attributes in a process of refining the final product model. Some tokens in an information menu may be never used to form information constructs. Nevertheless, a product model as a whole is not a subset of an information menu because additional constraints can be added to a product model later.

Forth, some of the semantic relations between tokens defined in an information menu will be inherited to a product model, but not all. If there are conflicts between the semantic relations, only the selected ones will remain. Also the relations can be changed through a normalization process.

Fifth, an aggregation of information constructs used in a process is not a *view* of an information menu. Since an information construct is a concatenation of tokens, it is obvious that an aggregation of ICs is not a subset of an information menu. However, it is open to further discussion whether an aggregation of ICs is a semantic derivation from tokens or not.

Table 4.2 Information constructs and entities in a product model

Information Constructs	**Entities in a Product Model**
PIECE+MATERIAL*CONCRETE{strength}	ENTITY piece material: material; END ENTITY; ENTITY material SUPERTYPE OF (concrete); strength: REAL; END ENTITY; ENTITY concrete SUBTYPE OF (material); END ENTITY;

The structure and the relations between information categories of RCM are summarized in Figure 4.23.

An information set is a set of information items. Information items are categorized into two types: information constructs (ICs) and vernacular information items (VIIs). ICs and VIIs have a mapping relationship. Information constructs is composed of several tokens. Tokens are categorized by general knowledge representation (KR) concepts and also by the entity/attribute distinction.

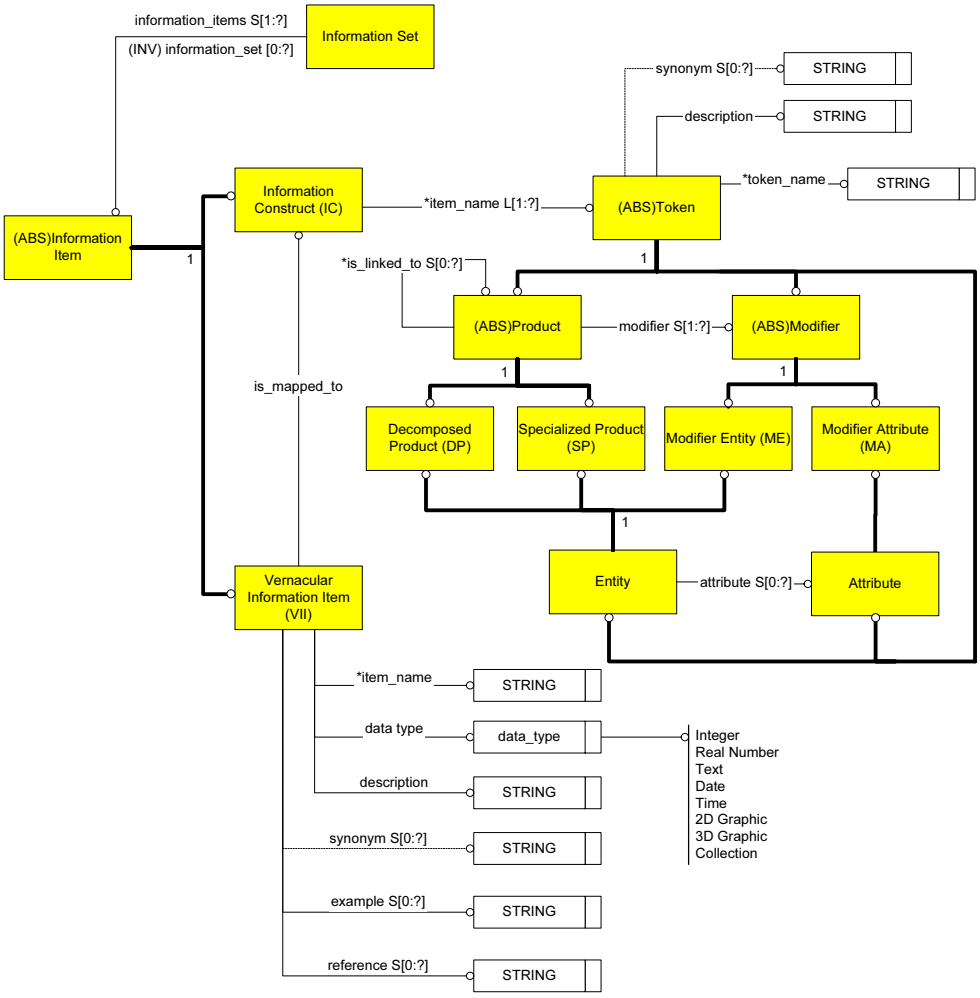


Figure 4.23. Information structure of RCM

VIIs are local terms of a company or a certain group of people. Each IC and VII should be unique. VIIs can have many synonyms. Examples, references, descriptions, and data types of VIIs should be provided for later mapping between ICs and VIIs. The following shows an example of a VII "delivered date" and its attributes:

Name: *delivered date*

Date Type: *date*

Description: *date when a piece is delivered to a site, NB:*

It may be different from shipped date.

Synonyms: *(empty)*

Example: *Oct 5, 2001*

Reference: *Packing slip*

4.8 DYNAMIC CONSISTENCY CHECKING

The quality of information generated by domain experts in the requirements collection phase is an important determinant of the quality of the resulting data model because RCM is based on process model information. Thus, a rigorous method to validate a model and its information flows is a key to its success. This section introduces the logic of consistency checking using information flows, and describes how it helps modelers to automatically and dynamically validate their models in real-time.

Any process-modeling method must rely on *semantic validation* and *syntactic validation methods*. In semantic validation, the only way in which modelers can confirm the consistency of a model is by considering what information is necessary for an activity or in what order activities should be laid out. Semantic validation methods are difficult to automate because the judgment often relies on domain-specific (sometimes case-specific)

experience and knowledge, which are difficult to generalize and transform into logic. Automated *syntactic validation* is available in most graphical modeling languages including the UML (Rosenberg and Scott 1999) and Petri-Nets (Eastman 1999). They check the consistency of a model subject to the syntax of their graphic symbols. What distinguishes RCM from other methods is that it incorporates the logic of checking *the consistency of information flow*, based on the interaction and interdependence of the activities with regard to the availability or unavailability of information: i.e., information used by an activity must be provided by its precedent activities, otherwise the activity cannot be performed and the model is inconsistent.

4.8.1 Notation of Dynamic Information Consistency Check

As described earlier, GT PPM has functions to collect, store, edit, and analyze information used in each activity in a process. These allow modelers to input information used for each activity as they build a process model. Among GTPPM symbols, this section focuses on two types of process semantics at a high level, i.e., the activity and the information flow (Figure 4.24). The information flow will be simply called a flow in this section for convenience.

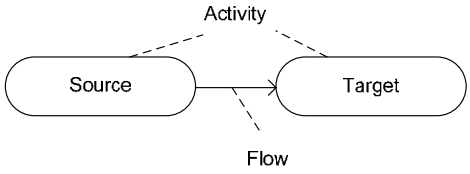


Figure 4.24 A source activity, a target activity, and a flow

The information used in each activity and flow is assumed to be collected and stored in each activity and each flow. Each flow has a single source activity and a single target activity (Figure 4.24).

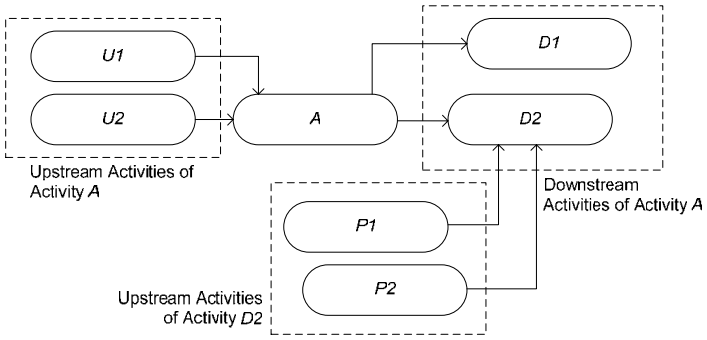


Figure 4.25 Upstream and downstream activities

Neighboring activities of an activity can be categorized ad hoc into *upstream* and *downstream activities* (Figure 4.25). A set of activities that provides information to an activity in a modeler's current focus is called a set of upstream activities. On the other hand, a set of activities that are fed with information by an activity in a modeler's current focus is called a set of downstream activities. In Figure 4.25, *U1* and *U2* are upstream information source of an activity *A*, and *D1* and *D2* are downstream activities of an activity *A*. Clearly, the definition of upstream and downstream activities is relative. *D2* can be called a downstream of *P1* and *P2*, and *P1* and *P2* can be called upstream of *D2*.

4.8.2 Basic Logic of Dynamic Information Consistency Check

The fundamental level of information consistency checking in GT PPM is an act of selecting and inputting information for a certain activity in a manner similar to the DFD method. If there is an activity 'Calculate the strength of a tire', a modeler may easily tell what information is necessary and what is not. We call this *semantic validation* of

information consistency. Still, there is no way to guarantee that information collected is complete or logically valid. Semantic validation is subjective because it is solely based on modelers' knowledge and judgment. For example, if 'engine volume' is an input information item for an activity 'Calculate the strength of a tire', a reader can guess that this is not right, but can only validate it by consulting a tire expert. Also, he/she cannot check if a critical variable is missing in the collected information. While semantic validation will always be partly a human responsibility, information consistency and robustness can be enhanced through logical checking.

The core concept developed here is called *validation by information dependency*: i.e., unless certain information is provided, other information cannot be generated. In the previous tire example (Figure 4.26), if 'engine volume' is provided as input to 'Calculate tire strength', we can infer that there is a certain dependency between 'the strength of a tire' and 'engine volume.' Conversely, if 'tire materials' is provided as output, 'tire materials' must be either input to or generated by 'Calculate tire strength.' By using this concept we can infer what information and Activities are missing.

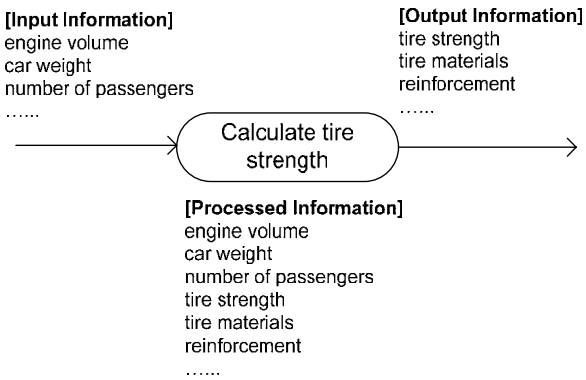


Figure 4.26 An example of "Calculate tire strength"

The first set of rules we initially implemented for checking information consistency was to compare the information set of an activity with the flows that stream into/out of the activity. The basic logic was that, by definition of information dependency, the information set of an activity must be an aggregation of information flowing into the Activity from source activities and the information generated in the activity itself. Therefore, a set of inflow information (F_u) must be a subset of information (I) of an activity (Figure 4.27 (1)). Conversely, any outflow information (F_d) must be a subset of its source activity (A) (Figure 4.27 (2)).

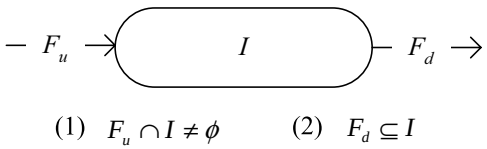


Figure 4.27 The basic logic

Expanding this logic, several rules are defined as follows (Figure 4.27):

Rule 8: Intersection of information (F_u) in any upstream flow that streams into a target activity and information (I) of the target activity (A) must not be an empty set:

$$F_u \cap I \neq \phi$$

Rule 9: A set of information (F_d) in any downstream flow that streams out of a source activity must be a subset of information (I) of the source activity (A):

$$F_d \subseteq I$$

Rule 10: By definition of a flow, a new information item (I_g) can only be generated in an activity (A) but not on any information flow (F). An information flow simply carries a set of information between activities:

$$\{i_g \mid (i_g \in I) \wedge (i_g \notin I_F)\}$$

where I: activity information; I_F: information of a flow

Figure 4.28 The first interface for checking the information consistency

The logic was initially implemented in GT PPM (Figure 4.28). This version was used for process modeling by the PCSC, which included 23 precast producers in the USA, Canada, and Mexico. Fourteen detailed models were collected and analyzed (Sacks et al., 2002). Even though the logic of this first approach was straightforward to understand, the information collected showed some inconsistency. It was found that, since it was very time-consuming and difficult for modelers to identify and report information for both

activities and flows, some modelers simply copied information from a flow to an activity without seriously considering the actual use of information. Moreover, since this logic was defined based on relations between information of an activity and its connected flow, relations between information in activities, which were our actual interests, could not be clearly shown. Therefore, I sought more rigorous definitions that could define relations of information between activities and that could validate information consistency between activities directly.

4.8.3 Extended Logic of Dynamic Information Consistency Check

The second approach focuses on relations of information within and between activities. In order to define information and its relationship more specifically, an information set of an activity is categorized into information input (I_i) and information output (I_o). The Information output is further subcategorized into passed through (without modification, I_{pt}), modified (I_m), and generated information (I_g) (Figure 4.29).

RCM categorizes information types into input and output information and subcategorizes them into five types. They are defined as:

- *Input (Information, I_i)*: Information required by this activity. Input is subdivided into:
    - *Remaining Information (Ir)*, which is purely referenced and is not transferred to the downstream activities and remains in an activity.
    - The rest of the Input Information:= Input (I_i) – Remaining Information (I_r)
- *Output* (Information, I_o): Information available from this activity.
    - *Information Modified (I_m)*, whose values are potentially changed or modified in this activity.
    - *Information Passed-Through (I_{pt})*, which is not modified by the activity, but transferred to the downstream activities as output.

- *Information Generated (I_g)*, i.e. newly generated in this activity.

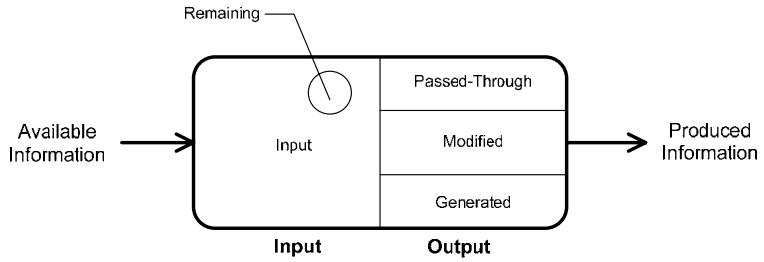


Figure 4.29 Types of activity information

Figure 4.29 illustrates the information types of an activity in RCM. Note that input information excludes information items returning through feedback flows in consistency checking. In addition to input and output information, references for checking information consistency are defined: i.e., *unavailable, unused, and not-provided information*. The relationships between information items imply *functional dependency*: i.e., input determines output and output is dependent on input. The rules that define the relationships between information types are:

Rule 11: Activity information set is the union of input information set and output information set:

$$I \equiv I_i \cup I_o$$

Rule 12: Output is the union of passed through, modified, and generated information:

$$I_o \equiv \bigcup\{I_{pt}, I_m, I_g\}$$

Rule 13: Input is the union of passed through, remaining, and modified information:

$$I_i \equiv \bigcup\{I_{pt}, I_r, I_m\}$$

Rule 14: The intersection of remaining, passed through, modified, and generated information is an empty set:

$$\bigcap\{I_r, I_{pt}, I_m, I_g\} \equiv \phi$$

Rule 15: By Rule 12 and Rule 13, remaining information is the subtraction of output information from input:

$$I_i - I_o$$

$$\equiv \bigcup\{I_{pt}, I_m, I_r\} - \bigcup\{I_{pt}, I_m, I_g\}$$

$$\equiv I_r$$

Rule 16: A set of Activity information is the union of input and generated information:

$I \equiv I_i \cup I_o$ (from Rule 11)

$I \equiv I_i \cup (I_{pt} \cup I_m \cup I_g)$ (by Rule 12)

$\therefore I \equiv I_i \cup I_g$ (by Rule 13)

Rule 17: Intersection of input (I_i) and generated information (I_g) is an empty set:

$\bigcap\{I_r, I_{pt}, I_m, I_g\} \equiv \phi$ (from Rule 14)

$I_i \cup I_g \equiv \phi$ (by Rule 13)

Thus far, we defined internal information types of an activity and their relationship. The relations of information between activities are redefined according to these new

information types and in-/out- flow information. The basic assumption is that the input information can receive information only from upstream activities, and the output information can provide information only to downstream activities. The relations are defined as follows (See Figure 4.25 and Figure 4.29 for reference):

Where

dn(A): downstream activities of an activity *A*;

up(A): upstream activities of an activity *A*;

output(A, x): output information of an activity *A*;

input(A, x): input information of an activity *A*

Rule 18: The input (I_i) of an activity (*A*) must be a subset of the unionized output (I_o^{U1}, I_o^{U2} I_o^{U3} … , I_o^{Un}) of its upstream activities (*U1, U2, U3…Un*):

$I_a \supseteq I_i$, where Available Information I_a

$\equiv \bigcup\{x \mid \text{output}(\text{up}(A), x)\}$

Rule 19: The output (I_o) of an activity (*A*) must contain the set of unionized input (I_i^{D1}, I_i^{D2}, I_i^{D3} … , I_i^{Dn}) of downstream activities (*D1, D2, D3…Dn*) of *A* less the set of aggregated output (I_o^{P1}, I_o^{P2}, I_o^{P3} … , I_o^{Pn}) of their upstream activities (*P1, P2, P3 … Pn*), excluding the activity *A*:

$I_o \supseteq I_{ro}$, where Required Output Information I_{ro}

$\equiv \bigcup\{x \mid \text{input}(\text{dn}(A), x)\}$

$- (\bigcup\{y \mid \text{output}(\text{up}(\text{dn}(A)), y)\}$

These ruless can be used for checking the consistency of information flows in complete models. However, in order to practically help modelers to build more robust models, we subcategorized the check results into several additional information sets, which could support real-time consistency checking as models are composed. These are called references. The references include (See Figure 4.29 and Figure 4.30):

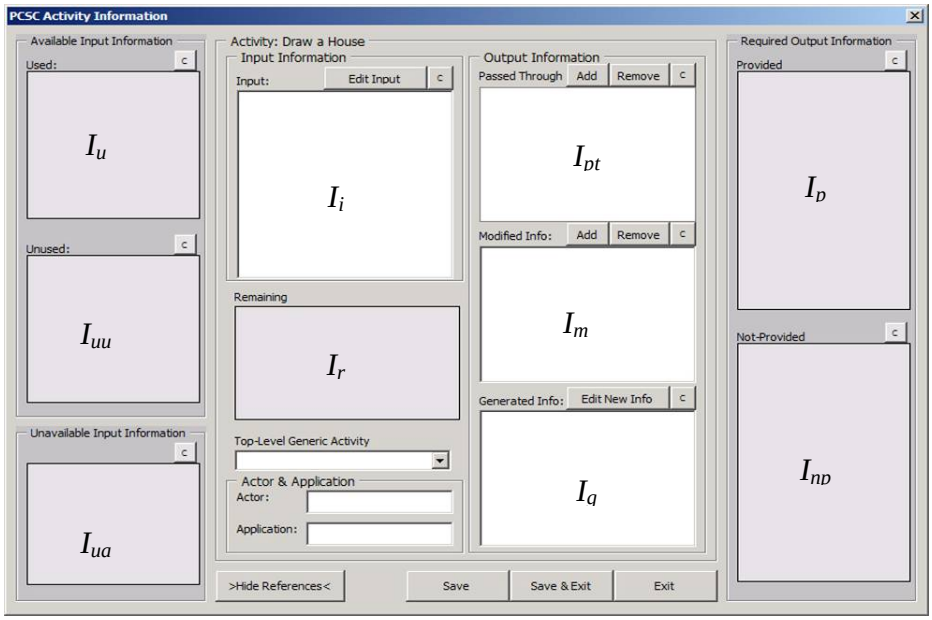


Figure 4.30 The second interface for checking the information consistency

Rule 20: A set of information that does not conform to Rule 18 is called unavailable information. In other words, input information that does not exist in available information is unavailable information:

Unavailable Information $I_{ua} \equiv I_i - I_a$

Rule 21: A set of information that is a subset of available information, but does not exist in input is unused information:

Unused Information $I_{uu} \equiv I_a - I_i$

Rule 22: Conversely, a set of information that is a subset of available information and also that of input is used information:

Used Information $I_u \equiv I_a \cap I_i$

Rule 23: A set of information that does not conform to $I_a \supseteq I_i$, where Available Information *Ia*

$\equiv \bigcup\{x \mid \text{output}(\text{up}(A), x)\}$

Rule 19 is *not-provided information* I_{np}:

Not-provided Information $I_{np} \equiv I_{ro} - I_o$

Rule 24: A subtraction of not-provided information from required output information is *provided information* I_p:

Provided Information $I_p \equiv I_{ro} - I_{np}$

The logical propositions are implemented in a user interface that automatically and dynamically checks the consistency of information as modelers edit a required output information list. Only input, passed through, modified and generated information are saved – the other categories are dynamically calculated based on these rules. As modelers update information, all the relations among relevant information sets are automatically rechecked and the check results are updated. In actual implementation, the derived rules such as Rule 15, Rule 16, and Rule 17 reduced the extent of source codes.

4.8.4 Practical Refinement of the Extended Logic

In this section, a practical refinement is introduced. While the extended logic of checking information consistency is theoretically robust, the interface shown in Figure 7 suffers the following drawbacks:

Selecting information items requires much work. In order to achieve a complete set of information for an activity, users must carefully and thoroughly think out what information is needed for four categories; i.e., for input, passed through, modified, and generated information.

- A process model of a medium-size organization usually includes hundreds of Activities for which information must be selected from a data dictionary (Sacks et al., 2002). In the case of the PCSC, a data dictionary with over 30,000 possible combinations of information is provided. Selecting the correct information from them for each Activity is not trivial. Modelers are apt to lose concentration and that can lead to an imprecise model. Thus, selecting and editing information from a data dictionary should be reduced as much as possible
- These drawbacks can be reduced by implementing the extended consistency checking logic as follows:
- Passed through, modified, and generated information lists are merged into an output information list. In the extended logic, the author distinguishes passed through, modified, and generated information explicitly. However, generated information does not denote the information item that is first generated in the whole sequence, but locally generated information within an activity. That is, if

a user wants to keep track of the first activity in which an information item is generated, then information items should be considered in the context of the whole sequence, not that of an individual activity. An information item that does not appear in any previous activities is true newly-generated information, while an item that exists in the previous activities, is true modified or passed through information.

- In all cases, input information is drawn directly from the output of the upstream activities. Conversely, output information should provide any downstream information that is not fed by other activities. It would therefore be preferable to drag the information from that available or required rather than select it from a large data dictionary. This approach allows users to select and copy information from available information (I_a) and also from required output information (I_{ro}) based on Rule 18 and Rule 19. In this case, the information copied will be deleted from its source information list based on Rule 20 and Rule 24. For example, if a user selects information from an unused information list and add it to an input list, the added information will be removed from the unused information list automatically.
- Output information shares passed through and modified information with input information (Rule 12 & Rule 13). Therefore, information items in the lists of input and output information can be copied from/to each other. In this case, the copied information will remain in both input and output information lists.
- The lists of used and provided information are omitted so as to reduce the complexity of the user interface and to increase the viewing space for the

remaining other lists. Available, used, provided, and required output information types are hidden.

- Unavailable information (I_{ua}) can be derived by Rule 20. Any information that falls into the unavailable information list must be provided by the upstream activities. If no activity exists that can provide the unavailable information, a new activity (or activities) must be added and connected with an information flow symbol/shape.
- Required and not-provided information can be derived by Rule 19 and Rule 23.
- Not-provided information must either be added in the output information list, or it must be provided to the downstream activities from other activities.

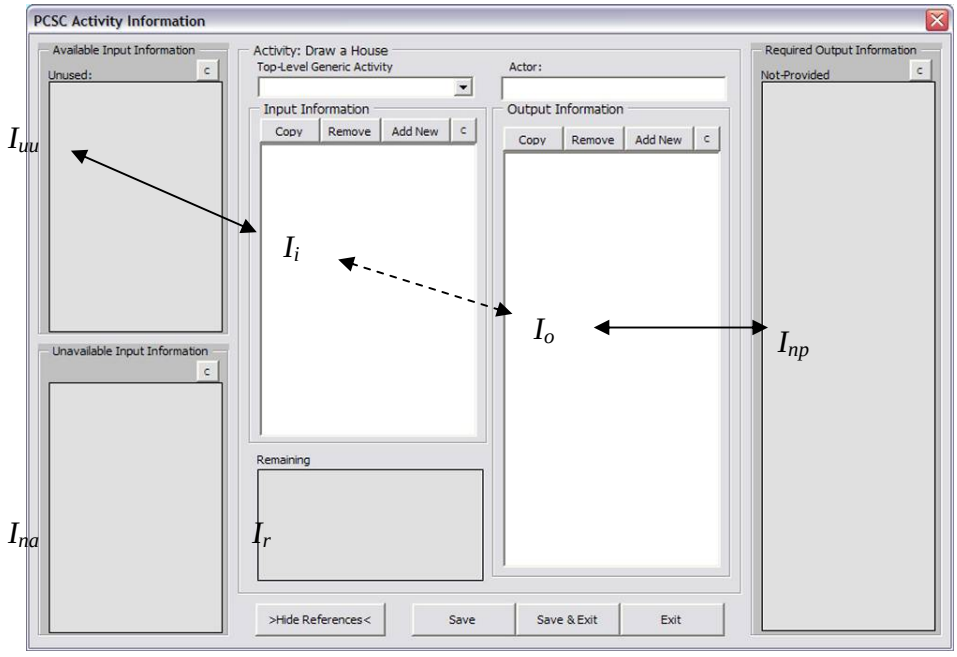


Figure 4.31 A practical approach to checking the information consistency

Since this new interface allows users to use selected information over and over through activities, it provides a degree of semantic consistency as well as reducing the

effort required of users in selecting information and maintaining the logical consistency between information collected.

4.8.5 Application & Limitations of the Dynamic Consistency Checking Method

It has been described how the consistency of information flows in a process model can be checked using logic based on information dependence between activities. A system can automatically check consistency of information flows according to this logic and can display inconsistent information items as *unavailable-* or *not-provided-information* lists. Modelers can maintain the consistency of a process model and its information flows by revising a model in any or all of three ways:

1) editing the information lists of relevant activities
2) adding or removing activities
3) creating, removing, or diverting information flows.

However, modelers should be aware of the limitations of this method as a consistency checker for process models:

- This consistency checking is not aimed at finding the most efficient form of information exchange (cf. DFD). Rather it reflects business practices and policy and, therefore, allows redundancy of information.
- It can guarantee a certain degree of completeness and robustness, but cannot guarantee absolute completeness of collected information because of the nature of modeling efforts. For example, even as a model is compiled, requirements may change.

4.8.6 Comparing RCM with Other Requirements Collection methods

Requirements of a UoD can be captured in a variety of formalisms. Common methods include Flowcharts, UML Activity Diagrams, Use Case diagrams, Data flow Diagrams (DFDs) (Osborne and Nakamura 2000), and IDEF0 (NIST 1993) schemas. Both Flowcharts (ANSI 1991) and Activity Diagrams (Booch, Rumbaugh, and Jacobson 1999) are limited only to capturing sequences of activities and are not able to describe the information used in a process. Use Case diagrams (Jacobson, Jonsson, and Overgaard 1992), which are a part of the UML methodology, define a set of sequences in which each sequence represents the interaction of the things outside the system (its actors) with the system itself (and its key abstractions) (Booch, Rumbaugh, and Jacobson 1999). They do not explicitly bring out the "information" hidden in the use-case notation. Data flow Diagrams (DFDs) (Osborne and Nakamura 2000) represent flows of information in a system using information flow symbols, processes, external entities (a.k.a. source/destination, sink), and internal data storages (often files). The whole set of DFDs consists of several levels of diagrams. The top-level DFD is called a context diagram. Details of information that is transferred between processes and data storages is described separately and called a data dictionary. However, DFDs do not show workflows, i.e., decisions or sequences of activities. DFDs capture information required for 'system' design, but do not describe information flows in a sequence of activities. It is important to capture information in a context of its use because information is often aggregated and decomposed in a data model depending on its use-cases (i.e., in what process it's used and stored) not on its system configuration (i.e., in what machine it's used and stored).

IDEF0 (Integration Definition of Function Modeling (International Organization for Standardization 1994)) is a Federal Information Processing Standard (FIPS) supported

by ISO It is based on SADT (Structural Analysis and Design Techniques) and is designed to define the "functions of a system or subject area with graphics, text and glossary (NIST 1993)". As in DFD modeling, IDEF0 models have a hierarchical structure and take a top-down approach. A unique feature of IDEF0 is its ICOM codes (Input, Control, Output, and Mechanism arrows). Input and Output arrows indicate the data and object flows into and out of a function. Control arrows denote the "required conditions for a function," and Mechanism arrows represent the "means of performing a function". Arrow types (or flows) are categorized in terms of use, but individual information items are not carried by the arrows between functions. Detailed information can be defined separately in IDEF1x (or IDEF1), but there is no direct link between the two modeling techniques. These modeling methods are available in commercial tools (e.g., BPR®, Arena®, Rose®, Visio®, and SmartDraw®).

Table 4.3. RCM and other modeling methods.

RCM Components	UML Activity Diagrams	IDEF0	Flowchart	DFDs
Internal/External Activity or Function	×	×	×	○
Hierarchical Structure	×	○	×	○
Information Flow	○	○	○	○ (Data flow)
Feedback Flow	×	×	×	×
Material Flow	△ (Object Flow)	×	×	×
Continue Shape	×	×	×	×
Decision Shape	○	×	○	×
Static Info Source	×	×	○ (Storage)	○ (Source/Sink)
Dynamic Info Repository	×	×	○ (Storage)	○ (Files)
Information Menu (Data Dictionary)	×	×	×	△

○: available; ×: not available; △: similar

The differences in modeling concepts between the RCM and other modeling methods with similar purposes are summarized in Table 4.3. (Use-Case diagrams are not included in Table 4.3 because they do not have conceptual commonality with the RCM except for the fact that both methods focus on use cases). In Table 4.3, the data dictionary of DFDs is marked with a triangle because it simply has the form of a collection of word-cards and does not have any structure or any method to deal with the large number of information items that can occur in a data model. Object flows in the UML are also marked with a triangle. They are somewhat similar to material flows in RCM; however they differ from material flows in that material flows are only restricted to physical materials while object flows include also non-physical objects and forms such as orders and bills (Booch, Rumbaugh, and Jacobson 1999).

Some commercial CASE (Computer-Aided Software Engineering) tools for database design (such as Visio®, AllFusion® (a.k.a ERWin®, BPWin®), and Corporate Modeler®) are capable of coupling DBMSs mostly only with ARMs (e.g., IDEF1x, EXPRESS-G, the UML as a modeling package, and ER diagrams) and sometimes with process models (AAMs). Several other methods have been researched and developed: e.g., PetriNet (Benwell, Firns, and Sallis 1991; Petri 1962), OSMOS (Wilson et al. 2001), GPP (Wix and katranuschkov 2002), ISTforCE (Wix and Liebich 2000), ATLAS (Tolman and Poyet 1995), VEGA (Bakkeren et al. 1996), ICCI (Katranuschkov et al. 2002), PISA(Bakkeren et al. 1996)). Among these, PISA directly interrelates process and product modeling by assigning additional symbols to each ICOM (Input, Control, Output, Mechanism) code of IDEF0. Table 4.4 compares PISA and RCM. The others are mostly focused on workflow management methods. Including PISA, the author is not aware of

any formal method that can elicit discrete information items from heterogeneous business environments step by step, validate collected information items, and integrate them into an industry-level product model.

Table 4.4. Comparison of PISA and RCM

	PISA	**RCM**
Basis for process modeling	IDEF0	N/A
Basis for product modeling	NIAM (graphical) EXPRESS (textual)	Information Menu EXPRESS (final result)
Integration of process and product modeling	Yes	Yes

CHAPTER 5

LOGICAL PRODUCT MODELING (LPM)

5.1 INTRODUCTION

The goal of LPM is to (semi-)automatically derive a product model from collected information constructs. The LPM is a combinatorial process of integrating and normalizing (i.e., decomposing and restructuring) information constructs into a formal product model. The targeted data modeling language, in this thesis, is EXPRESS. This chapter first discusses the data integration method and then the normalization method in LPM. Nevertheless, the integration and normalization processes are reciprocal and cannot be treated separately. The integration and normalization rules are also defined as *design patterns*[18]. A product model developed through GTPPM is by no means complete. Much of the information that provides the semantics (i.e., roles, rules, cardinalities, data types) still has to be added manually and the resultant model should be modified. The limitations are discussed at the end of this chapter.

5.2 SCHEMAS MAPPING, INTEGRATION, DESIGN PATTERNS, AND NORMALIZATION

Sometimes different work processes use the same set of data. Sometimes two equivalent processes may use the different sets of data. But different work processes usually require (and use) different sets of data. In order to make a product data model

[18] See the next section for more information on *design patterns*.

support various work processes, a product model should be an *integration* of different sets of data, which are required by different processes. We regard *integration* to be different from simple *aggregation*. While *aggregation* is a simple consolidation of data, *integration* is a semantic union[19] of different sets of data. In an integration process, semantic relations between different sets of data should be defined and mapped. Conflicts[20] between information constructs should be resolved.

There are several ways of mapping and integrating (sub-) schemas. A brief and general introduction to the EXPRESS integration method is available in Section 6.3 of (Schenk and Wilson 1994). Schenk and Wilson defines *integration* as a process of combining *Topical Information Models (TIMs)*, which are domain-specific information models developed by several modeling teams, into a minimally redundant, non-ambiguous and complete *Integrated Information Model (IIM)*. The TIM and the IIM are conceptually similar to the ARM and the AIM of the STEP method except for the fact that an IIM does not have predefined integrated generic resources (IRs). Schenk and Wilson categorize model integration into six forms:

1) *cosmetic integration*: modeling and documentation in a consistent style
2) *editorial integration*: elimination of synonyms and homonyms
3) *continuity integration*: elimination of redundancies and identification of gaps
4) *structural integration*: generalization of underlying concepts in TIMs and interfacing of an IIM with other IIMs
5) *core-based integration*: integration of TIMs into a high-level, abstract, and generic *core information model*

[19] See Appendix E for a formal definition of the semantic union.

[20] See the design patterns in the following sections for examples.

6) *evolution-based integration*: development of an IIM by integrating a TIM and another TIM until all the TIMs are integrated.

However, these descriptions provide only general guidelines and strategies for model integration and do not deal with the integration problems in detail.

Another effort worthy of discussing, is EXPRESS X. EXPRESS X is the ISO STEP schema mapping language that provides a formal description method to map two different schemas and their entities using *Rule Declaration* and *Type Map Declaration* (ISO TC 184/SC 4 1999). However, EXPRESS X is a mapping mechanism between two schemas, not an integration method.

For some reasons, many people misconceive that XML can automatically integrate two or more different schemas. XML has *Include, Import,* and *Redefine* mechanisms to reuse or to integrate different schemas into a new one (Wyke and Watt 2002). But they are not very different from the concept of the schema interfacing (or referencing) mechanism in EXPRESS. Both XML and EXPRESS provide *tools for model integration*, but the *work* still has to be done by humans. There is still no logic to automatically integrate two schemas in XML and EXPRESS, which the author is aware of.

Another approach to mapping and integrating (sub-) schemas is to use *design patterns* in object oriented programming. Design Patterns originated from Christopher Alexander's Pattern Language for buildings and towns (Alexander et al. 1997). The design pattern for object oriented programming has grown as a new field through years of efforts by design pattern groups and conferences (e.g., Ward Cunningham and Kent Beck (Coplien 1999), Erich Gamma and his colleagues (Gamma et al. 1994), Pattern Languages of Programming conference (PLoP), and Object Oriented Programming conference

(OOPSLA), and Object Management Group (OMG)). Each design pattern describes a particular object-oriented design problem; the core of the solution; and the constraints, consequences, and trade-offs of its use. Gamma et al. categorized design patterns into three: Creational, Structural, and Behavioral Patterns in their book (Gamma et al. 1994). The first two categories deal with instantiation and composition/decomposition of objects. The last category deals with encapsulation of algorithms.

The design pattern approach can be also applied to normalization of a data model. *Normalization* is an activity of using the known semantics of data in the form of dependencies that may be a cause for potential "update anomalies" requiring unnecessary duplicate work as well as causing potential inconsistencies in a database. Normalization of data was first proposed by Codd (Codd 1972) in the context of the relational model. The process of normalization can be defined by constraints or conditions that must be satisfied progressively to achieve a higher "quality" or "goodness" of design (Elmasri & Navathe, 2004). The process successively decomposes the relations so that, after each decomposition, a higher normal form is met; yet, the decomposition must be "non-additive" – in that it does not produce any spurious data after joining the component relations. The relational normalization theory is well accepted and defines the well-known first through fifth normal forms considering functional, multi-valued and join dependencies. However, in practice, the higher normal forms like the fourth and fifth normal forms are rarely used because their dependencies are hard to detect or for performance reasons, so as to avoid joins. It is difficult to apply the conventional normalization criteria and dependencies to several application domains: e.g., the human genome databases (Kogelnik et al. 1998)).

However, in object-oriented data models, redundancy of data is less of a concern because of their efficiency of representing the specialization relationship and other relations using "pointers" compared to the relational data model, which relies on foreign key - primary key relationships. There have been several efforts to explore and develop different normal forms for object-oriented data models from relational normal forms (Beeri, Bernstein, and Goodman 1978; Tari, Stokes, and Spaccapietra 1997). They illustrate that object-oriented models can be decomposed and integrated relatively freely depending on the given normalization criteria. However, unlike relational normalization, the goals (or criteria) of normalization are not clearly set in object-oriented data modeling languages. For example, (Tari, Stokes, and Spaccapietra 1997) proposed user-interpretation-based normalization. Three functional dependencies (i.e., path dependency, local dependency, and global dependency) were provided to support the method. Even though their method supports normalization (restructuring) of objects by user-defined constraints, the method is weak in terms of providing a standard or generic normalization method because any user-defined constraints can be a “norm.”

The following sections describe a method to integrate collected information constructs into an Application Requirements Model (ARM) and define *design patterns* to resolve conflicts between different information constructs in the collected information requirements and to normalize an integrated model.

5.3 INTEGRATION OF COLLECTED INFORMATION IN GTPPM

In GTPPM, there are four possible integration approaches. Figure 5.1 illustrates the four options. The bold line indicates a point of integration. The double-lined circle indicates an integrated model or an aggregated model.

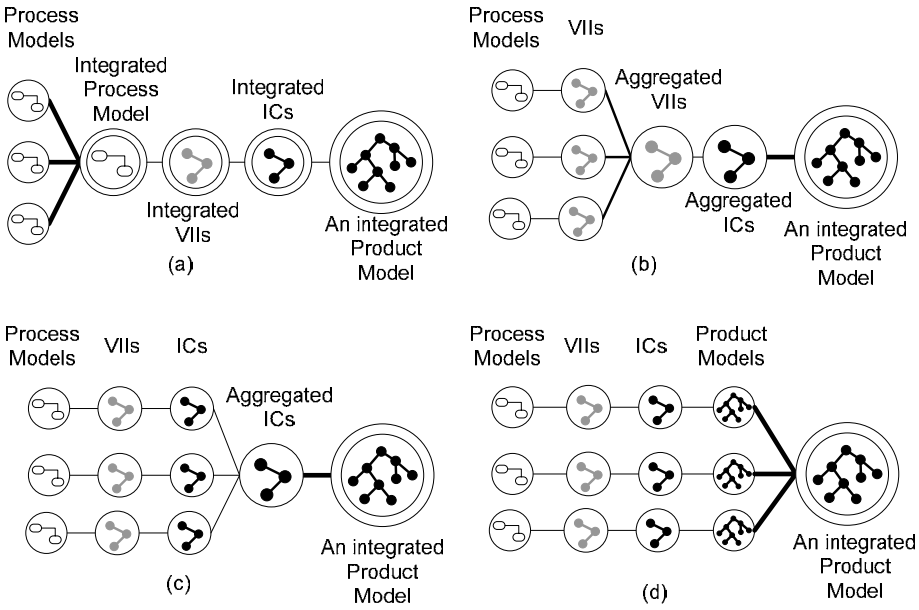


Figure 5.1 Four possible information integration methods in GTPPM

a) *Integration of process models*: Different process models can be integrated into one process model. Information requirements will be defined based only on a single unified process model. This is the most common approach that is taken today by STEP and IFC modeling efforts.

b) *Integration of vernacular information items*: Information items required by each process can be specified in each company's local terms. Specified vernacular information items (VIIs) can be aggregated and mapped to information constructs. Then the semantic conflicts can be resolved in the normalization process of information constructs (ICs) into a product model.

c) *Integration of information constructs*: Lists of information constructs can be aggregated into one large list. The aggregated ICs can be normalized into an integrated product model. The semantic conflicts should be resolved in the normalization process.

d) *Integration of data models*: A product model can be derived from each RCM model. And the generated product models can be integrated into a final product model.

Among these, this study takes the second and third approaches. In the first approach, if the process models can be integrated "losslessly", an integrated product model, which can support various processes, can be developed from the integrated process model. It will not be easy to integrate processes without losing any semantics. However, even if process models can be integrated losslessly, it will be difficult for modelers to specify information requirements based on an integrated process model because an integrated process model may not be able to represent the real contexts of information use.

The fourth approach is not very different from general schema integration. Each product model will have additional constraints (e.g., arities, rules) to the preliminary product models directly derived from RCM models. And the more detailed and complex constraints will be, the more difficult to resolve the conflicts between them. Thus, it is better to integrate information constructs when they are as little structured as possible.

In this regard, the second and third approaches are most feasible among the four possible integration approaches. The two approaches are interchangeable because a data structure is not sensitive to the *order of aggregation* (albeit it may be sensitive to the *order of integration*).

5.4 NORMALIZATION IN GTPPM

The definition of normalization in GTPPM is not very different from most existing ones (i.e., decomposition and restructuring of a data structure to a normal form), but the

scope and goals are not the same. Unlike traditional relational database normalization theories, the goal is not to eliminate redundancies or anomalies at an instance level (e.g., null value, lossless joint, multi-valued dependencies (Elmasri and Navathe 2004), but at an entity level. Since GTPPM is a schema generation method, the instance-level normalization issues are out of its scope. Also it does not deal with optimization issues that can make database transaction and query more efficient and faster. In any case, traditional database normalization and optimization methods can be applied when the final product model is implemented as a physical model.

The main goals of normalization in GTPPM are (1) resolving conflicts between information constructs with different data structures; and (2) eliminating redundant entities and attributes at a schema level.

5.5 LOGICAL PRODUCT MODELING IN GTPPM

As noted earlier, the integration and normalization processes are separated. Conflicts occurring in the integration process will be resolved during the normalization process. The current LPM is composed of eight steps.

Step 1: Union information constructs

Step 2: Decompose information constructs into entities by the association and decomposition relations.

Step 3: Detect and merge semantically equivalent entities.

Step 4: Detect and merge semantically equivalent attributes within entities.

Step 5: Resolve conflicts between attributes of a supertype and its inherited attributes.

Step 6: Generalize the data structure: Extract supertypes and their attributes from information constructs

Step 7: Resolve conflicts between attributes, supertypes, and subtypes.

Step 8: Refine the automatically derived product model

The LPM process is described in detail according to these eight steps in the following sections with examples and nine design patterns. The nine design patterns were defined to resolve the conflicts detected by MS SQL Server 2000® and EDM® through the evaluation process of three test cases described in Sections 7.3 through 7.5. Syntactically sound three test case models and an integrated model of the three test case models could be generated through the nine design patterns. We believe that these patterns are adequate for the normalization process as described. However, additional design patterns may be required and defined in the future.

5.6 STEP 1: UNIONIZING INFORMATION CONSTRUCTS

LPM first collects and unionizes the properties of all the information constructs in the RCM models. In this process, information constructs are unionized without any normalization or conflict resolution. Since the tokens of an information menu, on which information constructs are based, are defined based on the 'nym' principles (i.e., no homonym, no synonym), tokens with the same spelling should be regarded as identical. For example, if we have two information constructs of PROJECT, A and B, as shown below,

A: PROJECT ≡ {name, id, site}
B: PROJECT ≡ {name, manager, schedule, client}

the union of A and B will be:

$A \cup B$: PROJECT ≡ {name, id, site, manager, schedule, client}

The properties can be either a simple attribute or an entity in EXPRESS. This rule can be generalized as a design pattern. Note that the LPM process assumes that data types of attributes will be manually defined in the last of step of LPM in order to reduce the conflicts between attribute types. During the LPM normalization process, the data types of simple attribute types will be temporarily defined as STRING. And data types of entity-type attributes will be temporarily defined as the same as their *roles* (Figure 5.2) until manual modification. Also in the EXPRESS-G diagrams used for describing design patterns, roles will be omitted assuming that they are unique or the same as associated entities unless specified otherwise.

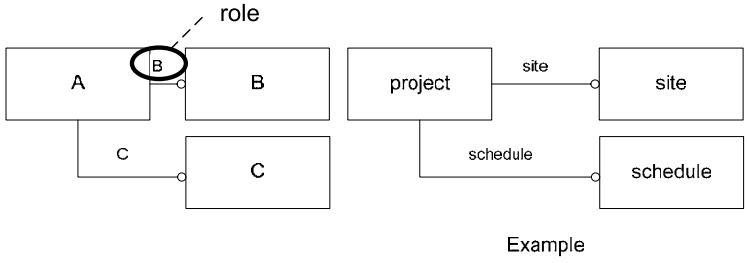


Figure 5.2 Roles of properties in GTPPM

In any case, if there are conflicts between data types, those should be resolved in the normalization process. (See Design Pattern 4 and Design Pattern 5 for examples and details on this issue.)

Design Pattern 1: Unionization of Information Constructs

Problem: Different information constructs denote that an entity has different (sets of) attributes (Figure 5.3) NB: Conflicting entities are in peach (or in grey in black and white print).

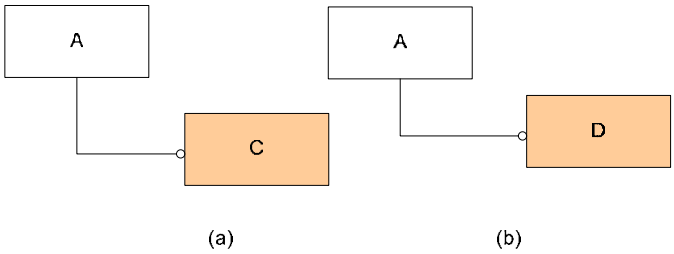


Figure 5.3 Conflicting attributes

Solution: Each relation has a specific meaning in structuring a data model. Thus, the general principle of model integration is to preserve semantics of information constructs as much as possible in an integrated model. Thus, the attributes of an entity should be the union of attributes of the entity defined in information constructs.

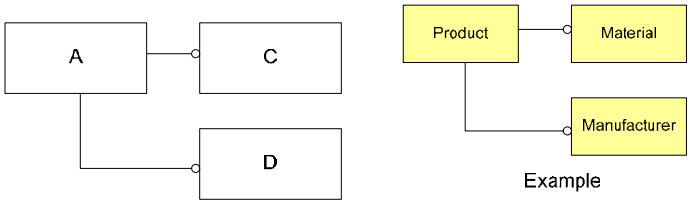


Figure 5.4 Unionization of Information Constructs

Notes: If there is a case that different attributes of an entity are associated with one entity type as shown in Figure 5.5 (a), there is no conflict in the information construct. An example is when A: schedule; Role_1: start_date; Role_2: end_date; and C: date. On the other hand, this should not occur if tokens are defined following the 'nym' principle, but if one attribute (name) is associated with two entities as shown in Figure 5.5 (b), one of the Role names should be changed unless there is a mechanism to integrate or merge the two attributes, namely, C and D in the example. An example is when A: product; Role_1: id; Role_2: id; C: unique_id; and D: design_model_id. In this case, either Role_1 or Role_2 should be renamed. Role_2 may be renamed to design_type.

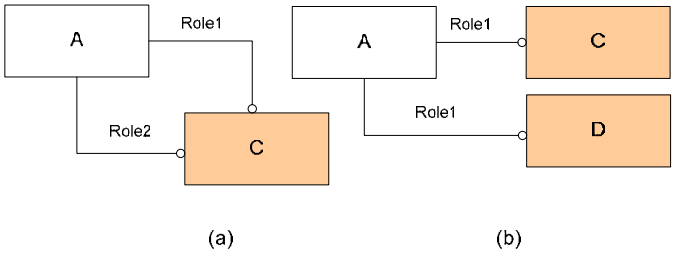


Figure 5.5 Conflicting attribute (role) names

Design Pattern 1 defines a situation when a property is shared by two "different" entities. It is also possible that a property is shared by two "different" entities. In such cases, the property should be regarded as a property of both entities.

Design Pattern 2: A Shared Entity Type

Problem: A property of Entity A (in Figure 5.6) is associated with Entity B. A property of another entity (Entity C), which is associated with a property of Entity A, is also associated with Entity B.

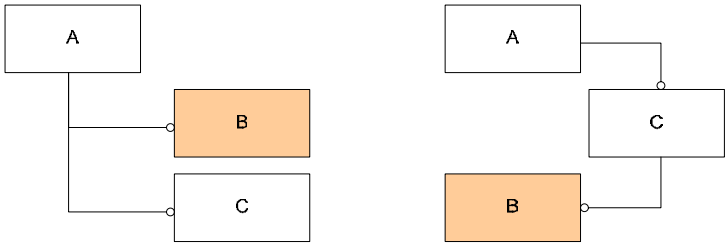


Figure 5.6 Properties associated with the same entity

Solution: Two different entities can have properties that are pointed to the same entity. Entities A and C in Figure 5.6 are two different entities. Therefore, there is no conflict in this case.

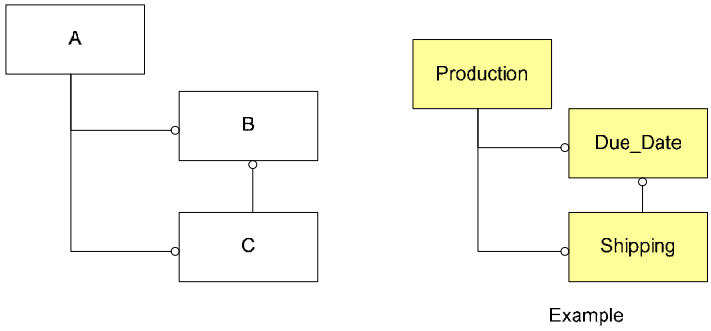


Figure 5.7 An example of two different properties associated with the same entity

5.7 STEP 2: DECOMPOSITION OF INFORMATION CONSTRUCTS

Information constructs are a concatenation of tokens linked by the decomposition relation and/or the specialization relation. Through the LPM process, the collected information constructs will be broken down either into entities or into attributes. LPM Step 2 is the first step to break down information constructs into smaller chunks by the association and decomposition relations. The decomposition procedure in Step 2 is as follows:

a) As described in Section 4.6.3, the decomposition/association relation is represented as "+" in GTPPM. If entities in an IC are concatenated by "+", then decompose ICs into separate entities. For example, an IC PIECE+GEOMETRY+DIMENSIONS{length} will be decomposed into three entities PIECE, GEOMETRY, and DIMENSIONS{length} in this process.

b) If an entity already exists, do not create a new one, but merge the attributes of the entity into the existing one following the Design Pattern 1. This is to avoid redundancy of entities.

c) If there is a concatenation of "A+B", the entity B should be added as an attribute of A. For example, GEOMETRY in the PIECE+GEOMETRY

concatenation should be added as an attribute of the PIECE entity. See Table 5.1 for more examples.

d) Step 2 should also conform to Design Pattern 1 and Design Pattern 2.

Table 5.1 illustrates an example of decomposing ICs in EXPRESS. *: denotes the specialization relation as described in Section 4.6.3.

Table 5.1 Decomposition of ICs with the decomposition/association relations

Information Constructs
PIECE+GEOMETRY+DIMENSIONS{length} PIECE+GEOMETRY*GEOMETRY_3D{volume}
Decomposition in EXPRESS
ENTITY piece geometry: geometry; geometry*geometry_3d: geometry*geometry_3d; END ENTITY;   ENTITY geometry dimensions: dimensions; END ENTITY;   ENTITY dimensions length: REAL; END ENTITY;   ENTITY geometry*geometry_3d volume: REAL; END ENTITY;

5.8 STEP 3: MERGER OF SEMANTICALLY EQUIVALENT ICS

Some information constructs are in different structures, but semantically represent the same thing. Step 3 identifies the semantically equivalent information constructs based on the *abbreviation rule* defined in Section 4.6.3. Examples are:

```
STAIRCASE{piece_mark, length, height, width}
ASSEMBLY*STAIRCASE{piece_mark, component_list }
PIECE*ASSEMBLY*STAIRCASE{assm_mark, num_of_steps}
STAIRCASE{piece_mark, num_of_steps, balusters}

PIECE*ASSEMBLY*STAIRCASE
≡ ASSEMBLY*STAIRCASE
≡ STAIRCASE                    (by the abbreviation rule)
```

ASSEMBLY*STAIRCASE represents the relationship between ASSEMBLY and STAIRCASE, but STAIRCASE alone cannot. Thus, the semantically equivalent entities should be merged into an unabbreviated form to capture as much semantics as possible. In the above example, the entities and attributes should be merged into PIECE*ASSEMBLY*STAIRCASE.

```
PIECE*ASSEMBLY*STAIRCASE {piece_mark, assm_mark, length, height,
width, component_list, num_of_steps, balusters}
```

This process can be generalized as Design Pattern 3.

Design Pattern 3 Merger of semantically equivalent entities

Problem: The specialization relation between two entities is represented in an information construct, but not in other information constructs.

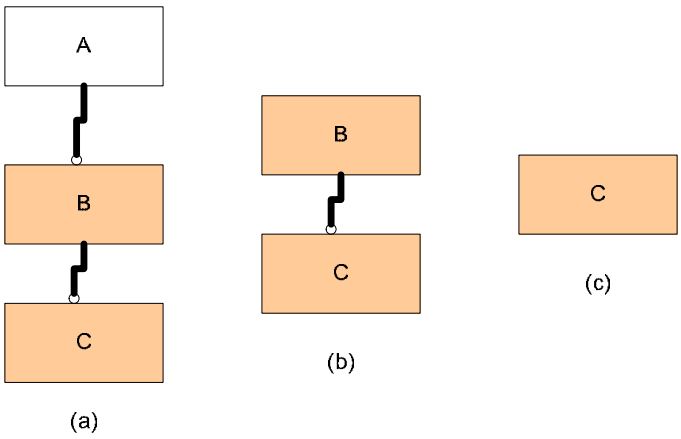


Figure 5.8 Semantically equivalent information constructs

Solution: Integrate the information constructs to the most semantically rich hierarchical structure.

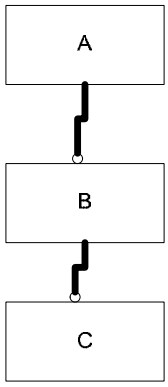


Figure 5.9 Merged entities in the specialization relation

A pseudo-code for detecting semantically equivalent ICs is provided in Appendix F.

5.9 STEP 4: RESOLVING CONFLICTS BETWEEN ATTRIBUTE TYPES

There can be a conflict between property (attribute) types. A property of an entity can be defined as an entity type in one information construct, but as an attribute type in

another. It can be also defined as STRING in one information construct and as an INTEGER in another. Design Pattern 4 and Design Pattern 5 deal with such conflicts between attribute types. Since the LPM process defines all the data types as STRING in the beginning, the second case does not occur. However, Design Pattern 5 provides a solution for such a case.

Design Pattern 4 A conflict between an entity type and an attribute type

Problem: A property may be defined as an entity type by one information construct and as an attribute type by another.

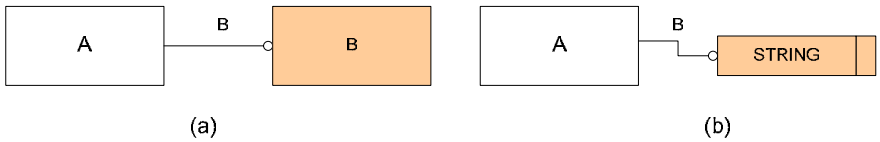


Figure 5.10 A conflict between an entity type and a simple type

Solution: An entity carries much richer information than an attribute type. Thus, the property should be defined as an entity. The order of selection should be:

Entity > User-defined types > Simple (attribute) types

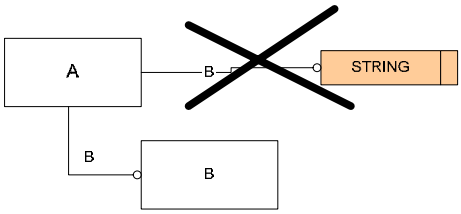


Figure 5.11 A resolution for the attribute data type conflict

Design Pattern 5 A conflict between simple attribute types

Problem: Different information constructs define the data type of an attribute as different simple types.

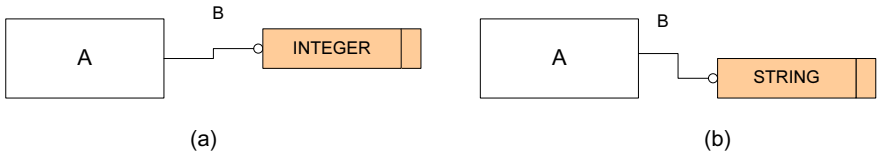


Figure 5.12 A conflict between simple types

Solution: The order of selection of simple types should be dependent on the inclusiveness of data types. For example, REAL can be expressed by STRING, but REAL cannot express STRING. LOGICAL can be expressed as 1, 0, -1 in INTEGER, but LOGICAL cannot express INTEGER. Thus, STRING is more inclusive than REAL. And INTEGER is more inclusive than LOGICAL. The order of inclusiveness of simple data types are:

BINARY > STRING > NUMBER > REAL > INTEGER > LOGICAL > BOOLEAN

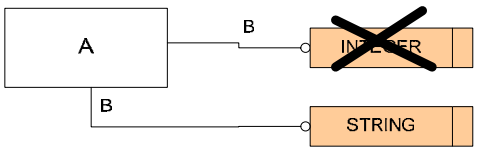


Figure 5.13 A resolution for the simple attribute data type conflict

5.10 STEP 5: RESOLVING CONFLICTS BETWEEN ATTRIBUTES OF A SUPERTYPE AND ITS INHERITED ATTRIBUTES

Design Pattern 6 Conflicts between attributes of a supertype and its inherited attributes

Problem: Attributes of a supertype are defined as attributes of its subtypes in different information constructs.

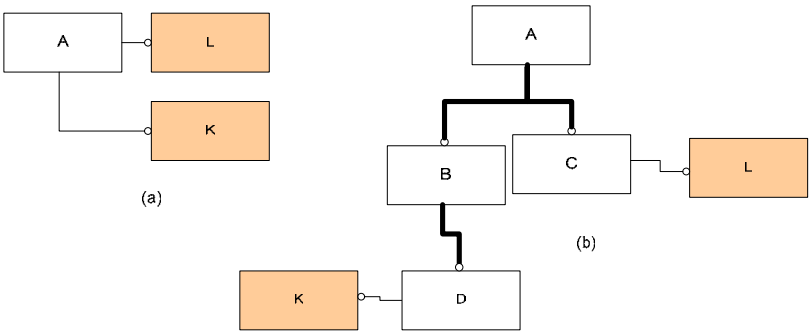


Figure 5.14 A conflict between attributes of a supertype and inherited attribute

Solution: Since all the attributes of a supertype will be inherited to its subtypes, it is redundant to define the attributes of a supertype again as attributes of its subtypes. The redundant attributes of subtypes should be deleted.

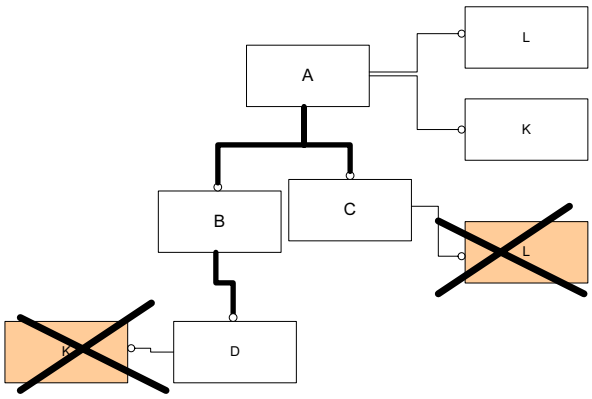


Figure 5.15 Deletion of inherited attributes

5.11 STEP 6: GENERALIZATION/SPECIALIZATION IN GTPPM

The goal of Step 6 is to restructure the entities by the specialization relation. The specialization relation is denoted as "*" in GTPPM as described in Section 4.6.3. Unlike tokens in the association/decomposition relation, tokens in the specialization relation cannot be simply decomposed and added incrementally because of the inheritance mechanism of the specialization relation. For example, if two subtypes B and C of a

supertype A have a common attribute D, the attribute D should be an attribute of the supertype A and should be removed from the subtypes B and C (Design Pattern 7).

Design Pattern 7: Generalization

NB: Design pattern 6 deals with a conflict between attributes of subtypes and attributes of their supertype whereas Design Pattern 7 defines a pattern for creating new attributes of a supertype by extracting least common attributes of its subtypes.

Problem: If subtypes of a supertype have common attribute(s),

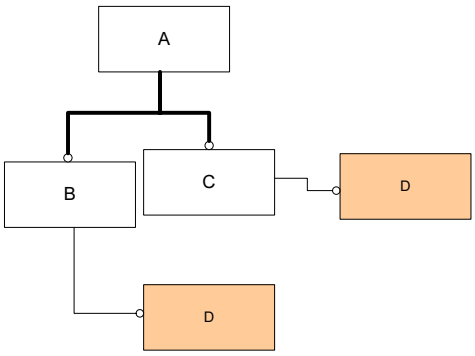


Figure 5.16 Common attributes of subtypes

Solution: the common attribute(s) should be deleted from the subtypes and added to the supertype.

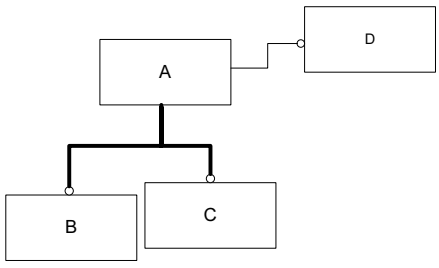


Figure 5.17 Generalization in GTPPM

Based on Design Pattern 7, a *supertype* can be formally defined as *a set of least common attributes of its subtypes*:

Supertype $T \equiv \{ x$: attribute; S: subtype of $T \mid \exists x \forall S(x \in S)\}$

Step 6 identifies and extracts *least common attributes* of subtypes from the collected information constructs and add them to their supertype. The extraction process must start with the top-level supertype because the top-level supertype is a set of the most common attributes of all the subtypes. For example, after Step 5, ICs may look like the examples below. At this point, there should not be any entities in the decomposition/association relation and any semantically equivalent items, which should have been resolved in the previous step.

```
piece*beam{piece_mark, length}
piece*wall{piece_mark, length, wythe}
piece*assembly*staircase{piece_mark, assm_mark, num_of_steps}
piece*assembly*facade{piece_mark, assm_mark, window}
```

First iteration: In the above example, PIECE is the top-level supertype. The most common attribute ‘piece_mark’ among the subtypes becomes an attribute of PIECE. PIECE and ‘piece_mark’ will be removed from the list. WALL, BEAM, ASSEMBLY will be marked as subtypes of PIECE.

```
~~piece*~~beam{~~piece_mark~~, length}
~~piece*~~wall{~~piece_mark~~, length, wythe}
~~piece*~~assembly*staircase{~~piece_mark,~~ assm_mark, num_of_steps}
~~piece*~~assembly*facade{~~piece_mark~~, assm_mark, window}
```

The PIECE entity in EXPRESS can be defined as follows:

```
ENTITY piece
        SUPERTYPE OF (beam, wall, assembly*staircase, assembly*facade);
        piece_mark: id;
END_ENTITY;
```

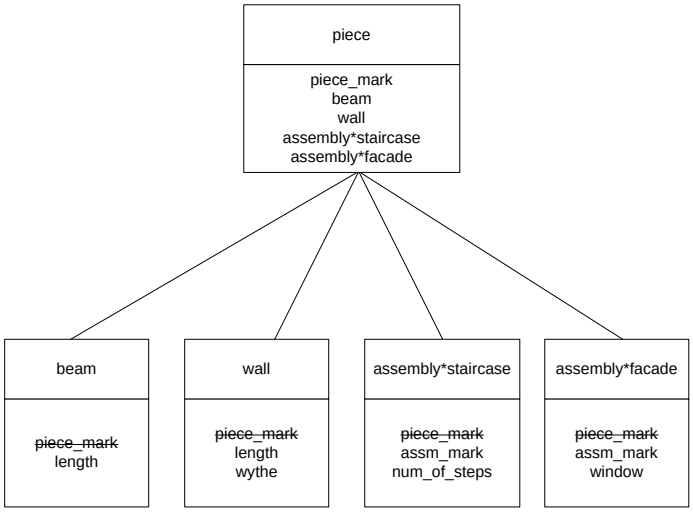


Figure 5.18. The first iteration of specialization

Second iteration: After removing PIECE from ICs in the first iteration, ASSEMBLY becomes the top-level supertype of STAIRCASE and FACADE. ‘assm_mark’ is the common attribute between them. A new entity ASSEMBLY is created, and ASSEMBLY and assm_mark are deleted from the list.

```
beam{length}
wall{length, wythe}
~~assembly*~~staircase{~~assm_mark,~~ num_of_steps}
~~assembly*~~facade{~~assm_mark,~~ window}
```

At this step, PIECE and ASSEMBLY can be described in EXPRESS as follows. Since ASSEMBLY*STAIRCASE and ASSEMBLY*FAÇADE have been decomposed, the two entities should be removed from the subtype list of PIECE. And ASSEMBLY should be added as a new subtype. Figure 5.19 is an illustration of the second iteration.

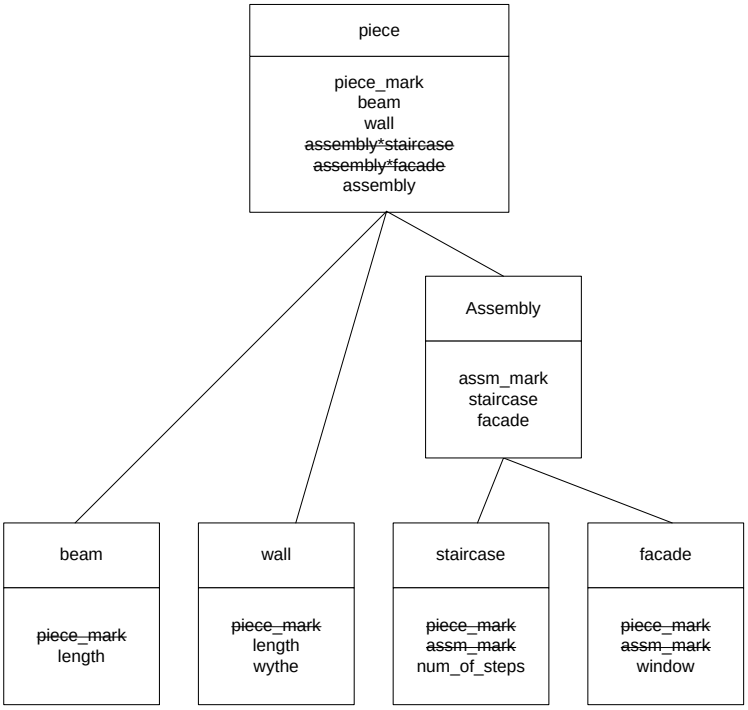


Figure 5.19. The second iteration of specialization

```
ENTITY piece
        SUPERTYPE OF (beam, wall, assembly*staircase, assembly*façade, assembly);
        piece_mark: id;
END ENTITY;

ENTITY assembly
        SUPERTYPE OF (staircase, facade);
        assm_mark: id;
END ENTITY;
```

The generalization process in LPM can be formally defined as a pseudo code and a design pattern as follows:

```
DIM supertype as ENTITY
DIM supertypes as SET_OF_SUPERTYPES
DIM subtype as ENTITY
DIM subtypes as SET_OF_SUBTYPES
DIM attr as ATTRIBUTE
DIM attrs as SET_OF_ATTRIBUTES
DIM name as ENTITY_NAME


SUB specialization
        DO WHILE exists(the_least_common_attrs)
                attrs = get_common_attr(subtypes)
                create_supertype(name)
                add_attr(attrs)
                add_FK(supertypes)
                delete_obsolete_FK(supertypes)
                delete_added_attrs(subtypes);
        LOOP
END SUB
```

5.12 STEP 7: RESOLVING CONFLICTS BETWEEN ATTRIBUTES, SUPERTYPES, AND SUBTYPES.

Design Pattern 8 Conflicts between a subtype and a property

Problem: An entity B may be defined as a property of another entity A in one information construct, but also as a subtype of the entity A in the other information construct.

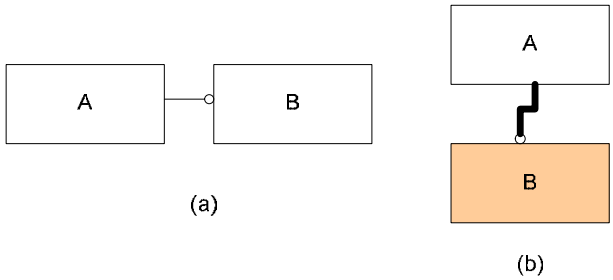


Figure 5.20 A conflict between an attribute and a subtype

Solution: An entity carries more information when it is defined as a subtype of the other entity than when it is defined as a property of the same entity because a subtype inherits attributes from its supertype. Thus, the entity should be defined as a subtype rather than as a property.

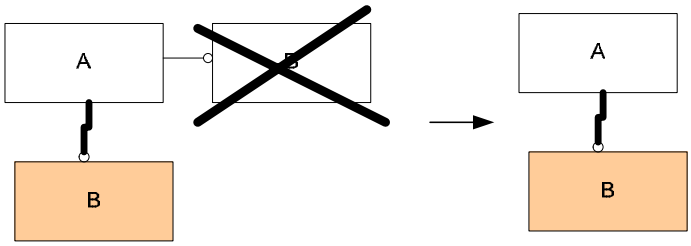


Figure 5.21 A resolution for the subtype and attribute conflict

If two additional information constructs in the specialization relation are added to the above example as shown in Figure 5.22, Entity C will be defined as a subtype of both Entities A and B, but Entity B will be also defined as a subtype of Entity A. Design Pattern 9 deals with such cases.

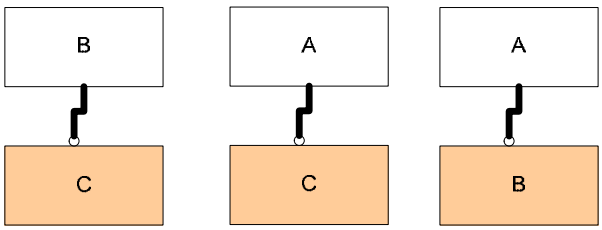


Figure 5.22 An additional IC

Design Pattern 9 A duplicate subtype relation

Problem: An entity is defined as a subtype of another entity twice: once directly from its supertype, the second time indirectly through another supertype.

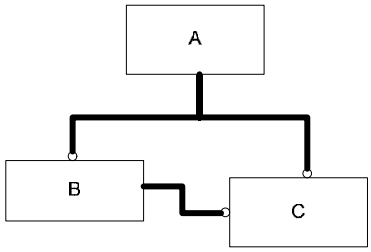


Figure 5.23 A duplicate subtype relation

Solution: Since all the attributes of a supertype will be inherited to a subtype through a hierarchical structure, it is redundant to define an inheritance relationship between a supertype and a subtype when the subtype is already linked to the supertype through a hierarchical structure.

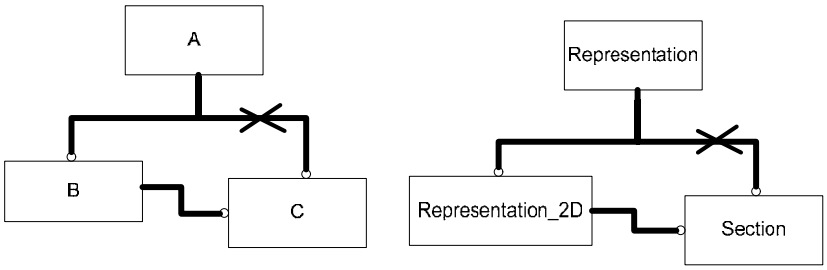


Figure 5.24 A resolution for a duplicate subtype relation

5.13 STEP 8: LIMITATIONS OF GTPPM & REFINEMENT OF A MODEL

Through steps 1 through 7, a syntactically sound EXPRESS model can be derived from collected information constructs. The following EXPRESS code is an example of automatically derived definition of `exterior_pc_column` through Steps 1 through 7:

```
ENTITY exterior_pc_column
    SUBTYPE OF (
    pc_column
```

```
    );
    surface_treatment: surface_treatment;
    clearance: string;
    hardware_list: hardware_list;
    rebar: rebar;
    rebar_cage: rebar_cage;
    pocket: pocket;
    corbel: corbel;
    geometry_3d: geometry_3d;
    foundation_drawing: foundation_drawing;
    elevation_drawing: elevation_drawing;
    detail_drawing: detail_drawing;
    geometry_2d: geometry_2d;
    plan_drawing: plan_drawing;
END_ENTITY;
```

By no means, the automatically derived product model is complete. For example, the current PIS system does not define *roles* and *cardinalities* defined in the beginning. For example, let's assume that the `BOM (bill of material)` entity in a final product model has an attribute `piece_list`, which is a list of `pieces`:

```
ENTITY BOM;
       piece_list: LIST [0:?] OF piece;
END_ENTITY;
```

Such semantics can be captured, but in a limited format using the current Product Information Specification (PIS) method:

```
BOM{piece list;}
```

And the automatically derived definition of `BOM` will be:

```
ENTITY BOM;
       piece list: piece_list;
```

```
END_ENTITY;
```

Thus, the automatically derived definition of BOM will not include any *cardinality* and *role* information. Such information should be added manually afterwards. In addition to the cardinality and role definitions, other data modeling semantics that cannot be captured by GTPPM and should be added after the LPM phase at this point are as follows. The distinction between mandatory vs. optional relations has not been made. Also the RULE, WHERE, DERIVE, and UNIQUE clauses have not been added. A product modeler may even remove or add entities or restructure some of the relations in. Also, simple (data) types (e.g., REAL or NUMBER) have to be redefined from the current STRING type. Other user-defined attribute types may need to be defined.

If a resultant product model is far from the expectation, the product modeler should re-examine the scope and activities defined in the RCM process models, the structure of the information menu, and information constructs defined within each activity.

CHAPTER 6

IMPLEMENTATION

6.1 AN ASSUMED MODELING PROCEDURE AND IMPLEMENTATION

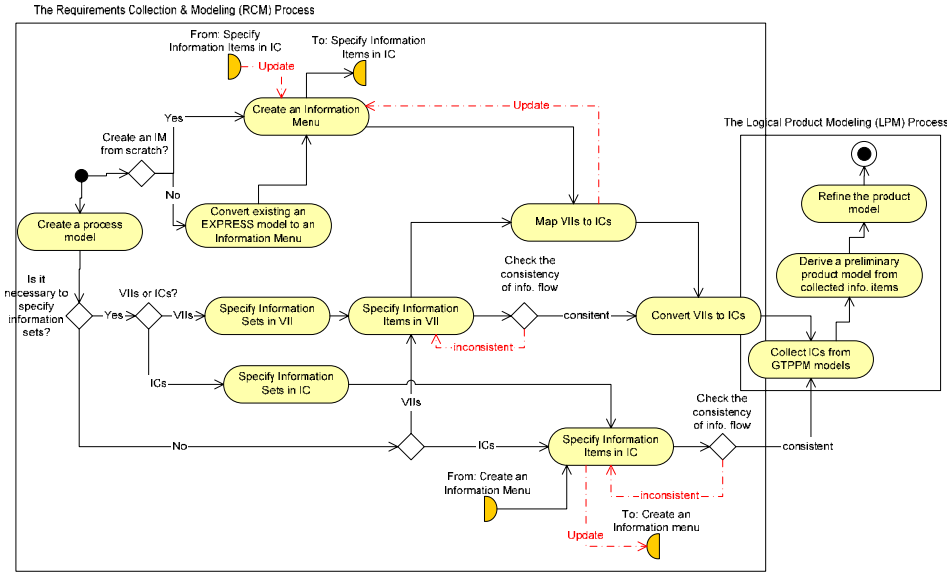


Figure 6.1 An assumed GTPPM modeling procedure

GT PPM has been implemented as a MS Visio® add-on. The tool is designed to support several modeling approaches illustrated in Figure 6.1. This chapter describes the modeling approaches and GT PPM interfaces to support each step in detail. GTPPM is a collaborative work process between domain experts and product modeling experts (mediators). Possible roles of domain experts and product modeling experts in each step are also described.

6.2 THE REQUIREMENTS COLLECTION AND MODELING (RCM) PROCESS

First, domain experts model a process without information such as examples shown in Figure 6.2. The GTPPM tool includes several functions to comply with the syntactic rules of process components defined in Sections 4.3 though 4.5. It can check disconnected flows, the direction (in/out) of flows, and the relations between high-level activities and their subsidiary detailed activities. It automatically generates identifiers for a pair of continue shapes and hyperlinks between them. Feedback flows in Figure 6.2 create loops through dynamic information repository (i.e., "Production Facilities") to other activities

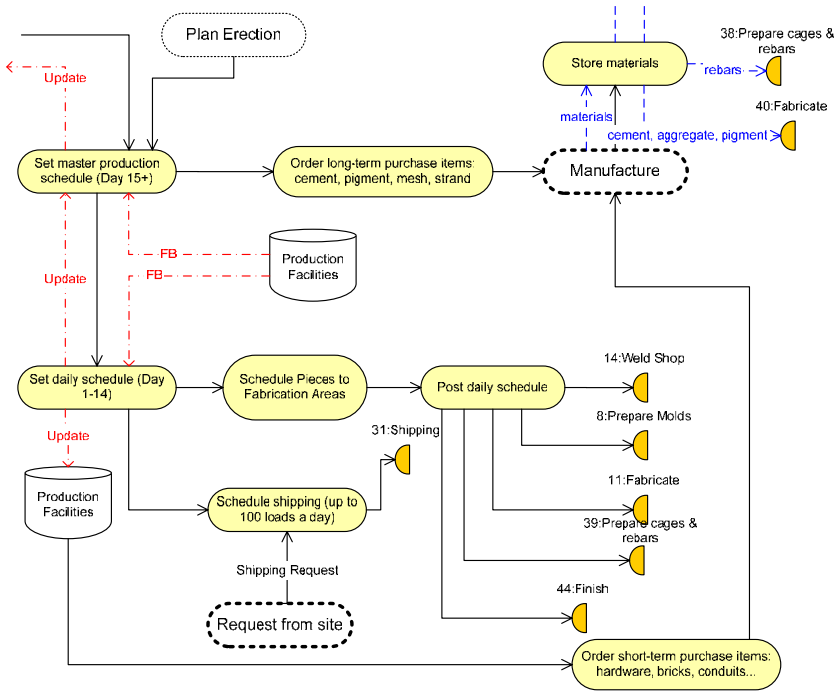
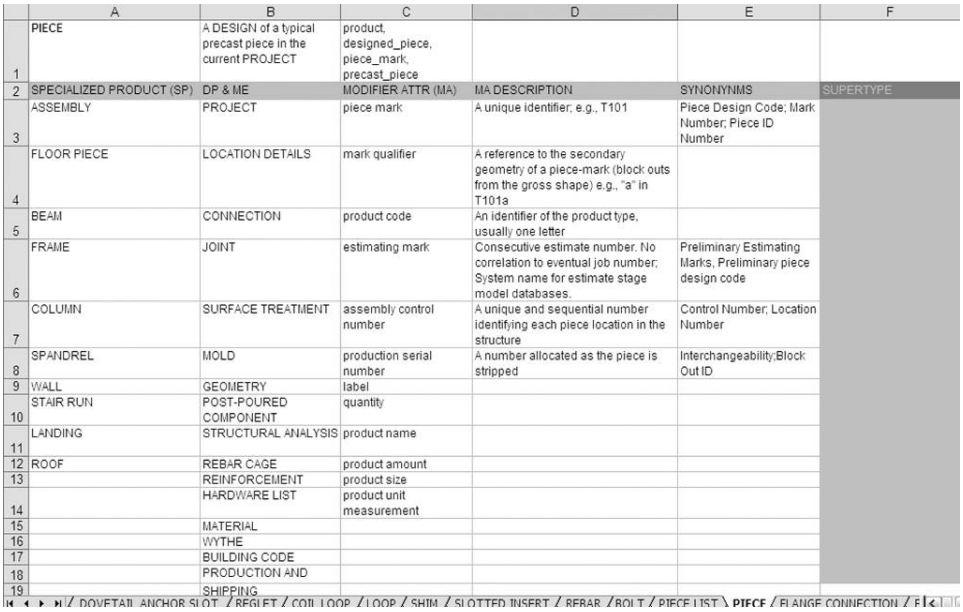


Figure 6.2. A part of a GT PPM model prepared by a precast concrete company

In parallel or in advance, product modeling experts (mediators) prepare an information menu (IM) in an Excel® file Figure 6.3. An IM includes a list of main products (PD) and the definitions of entities. Each entity definition specifies its specialized products (SPs or subtypes), decomposed products (DPs) and modifier entities (MEs), modifier

attributes (MA), and synonyms. Since GTPPM aims to derive an EXPRESS model, the specialization and the instantiation relations and the decomposition and the association relations are not distinguished. The SUPERTYPE field on the far right side of Figure 6.3 can be automatically filled using an IM (information menu) Macro. There are several other IM Macros developed to check the consistency of items defined in an information menu by checking misspelled entities and dangling entities (entities that are not associated with any other entities).



	A	B	C	D	E	F
1	PIECE	A DESIGN of a typical precast piece in the current PROJECT	product, designed_piece, piece_mark, precast_piece			
2	SPECIALIZED PRODUCT (SP)	DP & ME	MODIFIER ATTR (MA)	MA DESCRIPTION	SYNONYNMS	SUPERTYPE
3	ASSEMBLY	PROJECT	piece mark	A unique identifier; e.g., T101	Piece Design Code; Mark Number; Piece ID Number	
4	FLOOR PIECE	LOCATION DETAILS	mark qualifier	A reference to the secondary geometry of a piece-mark (block outs from the gross shape) e.g., "a" in T101a		
5	BEAM	CONNECTION	product code	An identifier of the product type, usually one letter		
6	FRAME	JOINT	estimating mark	Consecutive estimate number. No correlation to eventual job number; System name for estimate stage model databases.	Preliminary Estimating Marks, Preliminary piece design code	
7	COLUMN	SURFACE TREATMENT	assembly control number	A unique and sequential number identifying each piece location in the structure	Control Number; Location Number	
8	SPANDREL	MOLD	production serial number	A number allocated as the piece is stripped	Interchangeability;Block Out ID	
9	WALL	GEOMETRY	label			
10	STAIR RUN	POST-POURED COMPONENT	quantity			
11	LANDING	STRUCTURAL ANALYSIS	product name			
12	ROOF	REBAR CAGE	product amount			
13		REINFORCEMENT	product size			
14		HARDWARE LIST	product unit measurement			
15		MATERIAL				
16		WYTHE				
17		BUILDING CODE				
18		PRODUCTION AND				
19		SHIPPING				

DOVETAIL_ANCHOR_SLOT / REGLET / COIL_LOOP / LOOP / SHIM / SLOTTED_INSERT / REBAR / BOLT / PIECE_LIST \ PIECE / FLANGE_CONNECTION /

Figure 6.3 Entity PIECE defined in an Information Menu (IM)

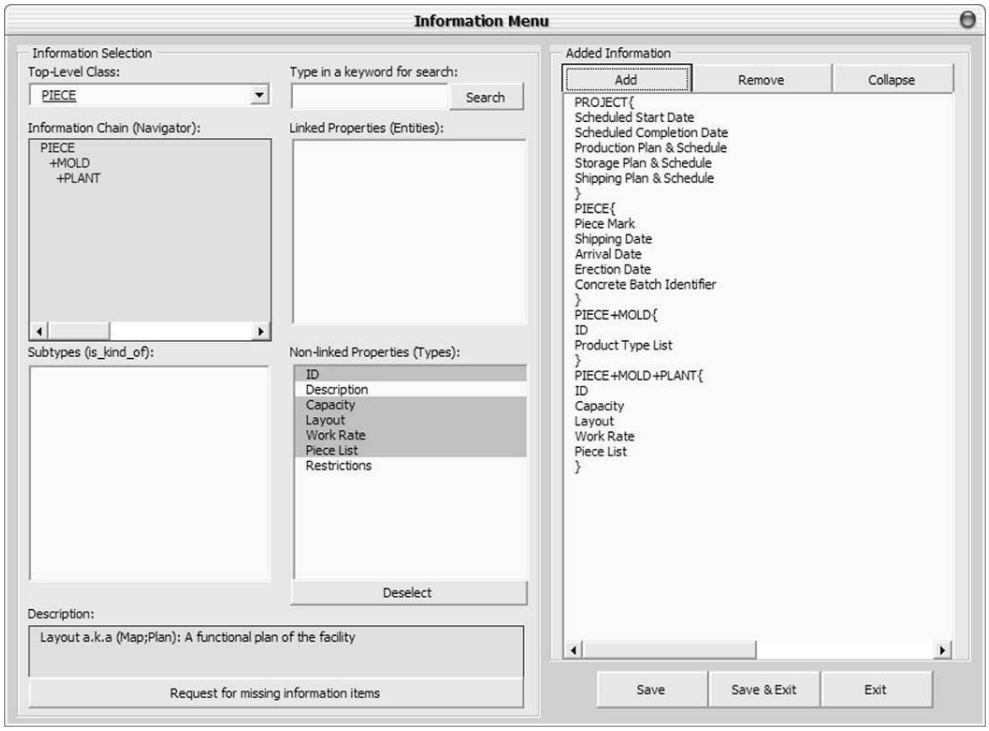


Figure 6.4. A GT PPM Information Menu Interface (the IC Editor)

Figure 6.4 illustrates the IC Editor. In a large project that includes heterogeneous business practices and domain experts with various experiences, expert modelers can create one or two GT PPM models as pilot models by visiting companies and generate an information menu. Most standard terms of an industry can be captured in this preparation phase. When an information menu is ready, other domain experts can join a modeling effort. Domain experts should map all their VIIs to corresponding information constructs (ICs) (Figure 6.12). GT PPM reads in an information menu from an MS Excel® file in real-time. Users can select, compose, and add information items (i.e., ICs) from the information menu to each activity. The left window of Figure 6.4 shows a hierarchical structure of tokens that represents aggregation, specialization, and classification (see Section 4.6). The right window shows ICs that are composed of tokens available from the left window.

While modeling a process, domain experts create a vernacular data dictionary (VDD). The vernacular data dictionary (VDD) includes information on information sets (Figure 6.5) and vernacular information items (VIIs) (Figure 6.6). It includes VII names, definitions, data type, examples, references, and synonyms.

	A	B	C	
1	Information Set			
2	Index	Name	Description	Information Items
3	1	PROJECT INFORMATION SHEET	A sheet that summarizes project information	{;project name;location;report date;purchaser;addre
4	5	PIECE DRAWING	piece drawing	##
5	8	ESTIMATE DETAIL	Estimate Detail w/ Crew Productivity & Manhours	##
6	12	ERECTION DRAWING	Erection Drawing, E-Drawing	{;revision date;revision by;revision no;drawn date;dr
7	17	STRAND SUMMARY	Strand Summary	{;project name;location;job#;estimate no;product na
8	18	SPECIAL PRODUCTION REQUIREMENTS	special productoin requirements for bid engineering	{;load condition;floor to floor height;fireproof require
9	19	DETAILED POUR SCHEDULE	Detailed Pour Schedule	##
10	20	DAILY CONCRETE POUR SCHEDULE	daily concrete pour schedule with dates	##
11	22	PACKING SLIP	Packing Slip, a.k.a. Bill of Loading	{;address;city_state_zip;job#;truck number;trailer nu
12	22	PIECE TAG	piece tag	{;bar code;piece weight;piece mark;}
13	25	BOM FOR HARDWARE	Bill of Materials for hardware	{;project name;job#;project phase;piece mark;draw
14	26	BOM FOR PIECE	Bill of Materials for piece	{;project name;job#;project phase;piece mark;draw
15	10	SCHEDULE	contractual project schedule	{;Contract Date;engieering date;review by architect;p
16	11	PROJECT REVIEW CRITERIA	project review criteria	##
17	12	JOB COST REPORT	job cost per product by man-hour cost	{;project name;location;project type;job#;estimate n
18	13	TAKEOFF LIST	takeoff list	##
19	2	PROJECT DIRECTORY	contact information for a speicific project;	{;contractor type;firm name;address;city_state_zip;c
20	3	SITE LOCATION	site location information	{;address;city_state_zip;jobsite phone;jobsite fax;}
21	4	FINANCIAL INFORMATION	financial information	##
22	5	ENGINEERING/DRAFTING REQUIREMENTS	contractual requirements for engineering, drafting	{;fire rating;load designer;sealed calculations;seale
23	6	JOB SUMMARY SHEET	job summary, take-off	##
24	7	MATERIALS REQUIREMENTS	contractual requirements for materials	{;Hardware/reinforcing included;Hardware/reinforcir
25	8	MIX/FINISH/SAMPLE INFORMATION	contractual requirements for mix/finish/sample	{;architectural mix/color ;structural mix/color;sample
26	9	FIELD RELATED SERVICES	contractual requirements for field related services	{;field related services included;field related service
27				

Figure 6.5 Information Sets defined in a Vernacular Data Dictionary (VDD)

Figure 6.8 illustrates the VII/VDD editor. A VDD is stored in a separate Excel® file[21] from an IM file. The VII name should be unique. If domain experts add a new item with the same name as an existing VII, it alerts users. The list of VIIs can get very long after a while. Domain experts can search for a term they defined by typing in part of the term. For example, if a domain expert types an unfinished word, e.g., “proj”, and executes “Search”, the VII editor search through VII names and synonyms and returns any terms with “proj” in their name and homonyms (Figure 6.8). Another core function of the VII editor is that, if domain experts want to update a term for some reasons (e.g., typos, a

[21] VDDs and an IM are stored in separate Excel® files because a VDD is only of interest of a certain modeler (team), but not of interest of the whole modeler teams. Only the IM will be shared by different modelers (or teams) and VDDs will be kept by each modeler (team).

conflicting name), it updates not only the term in the VDD, but also all the terms with the same name in the model (Figure 6.7).

	A	B	C	D	E	
1	Information Set Items					
2	Name	Description	Examples	References	Synonyms	Mapped Item
3	# of pours	number of pours	6	Strand Summary		PIECE+PRODUCTION AND H
4	# of strands	number of strands	60	Strand Summary		PIECE+REINFORCEMENT{;qu
5	address	street address		Project Inforamtion Sheet Page 1		PROJECT+SITE{;address;}
6	affidavits for stored materials	affidavits for stored materials	must be included	Project Information Sheet Page 2		PROJECT{;contract details;}
7	anchor bolt	anchor bolt				PIECE*COLUMN*EXTERIOR
8	architectural mix/color	requirements for architectural mix/co	tan	Project Information Sheet Page 3		PIECE+PRODUCTION AND H
9	area code	(project) area code	610	Take Off List		PROJECT+SITE{;address;}
10	average piece length	average poured piece length	43.00 feet	Detailed Pour Schedule		PIECE+GEOMETRY+CONST
11	average piece weight	average poured piece weight	46, 649 lbs	Detailed Pour Schedule		PIECE+GEOMETRY+CONST
12	bar code	bar code for a piece		Piece Tag		PIECE{;piece mark;}
13	batch finish time	batch end time	10:30 PM	Daily Concrete Pour Schedule		PIECE+PRODUCTION AND H
14	batch start time	batch start time	10:30 PM	Daily Concrete Pour Schedule		PIECE+PRODUCTION AND H
15	bent bar	bent bar		Unistress		PIECE*COLUMN*EXTERIOR
16	bent bar layout	bent bar layout		Unistress		PIECE*COLUMN*EXTERIOR
17	bent bar maximum spacing	bent bar maximum spacing		Unistress		PIECE*COLUMN*EXTERIOR
18	bidding timeframe vs. estimating sched	bidding timeframe vs. estimating sche		answers from David Bosch		PROJECT+ESTIMATION{;SC
19	billing forms required	billing forms required	AIA form req'd	Project Information Sheet Page 2		PROJECT{;contract details;}
20	block out	block outUnistress Corp Double Tee [	BODT-01: Double	Unistress Corp Double Tee Design Gu		PIECE+BLOCKOUT+3D GEC
21	bom checked by	BOM checker	DLC	Bill of Materials		PROJECT+DOCUMENTATIO
22	bom created by	BOM creator	DLC	Bill of Materials		PROJECT+DOCUMENTATIO
23	bond	included bond information	Not included	Project Information Sheet Page 2		PROJECT{;contract details;}
24	certificate of insurance	requirements for the certificate of ins	not required	Project Information Sheet Page 2		PROJECT{;contract details;}
25	certified payrolls required	certified payrolls required	no	Project Inforamtion Sheet Page 1	street address	PROJECT{;contract details;}
26	check result	check result		Unistress Corner Columns		PIECE*FLOOR PIECE*DT+QC
27	chord connection location	chord connection location: is required	Typically located	Unistress Corp Double Tee Design Gu		PIECE*FLOOR PIECE*DT+CC
28	chord plate design	chord plate design: is required to res		Unistress Corp Double Tee Design Gu	chord plate type	PIECE*FLOOR PIECE*DT+CC
29	chord reinf design	chord reinforcement design: is requir		Unistress Corp Double Tee Design Gu	chord reinforcement type	PIECE*FLOOR PIECE*DT+CC
30	chord reinf location	chord Reinforcement location: is requ	Typically located	Unistress Corp Double Tee Design Gu		PIECE*FLOOR PIECE*DT+CC

Figure 6.6 Vernacular information items (VIIs) defined in a VDD

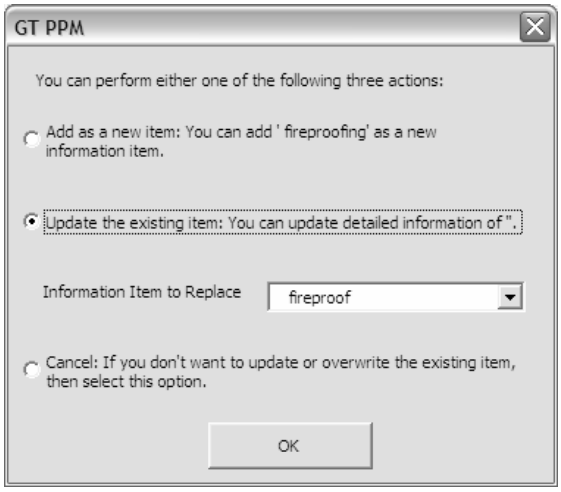


Figure 6.7 The VII Updater

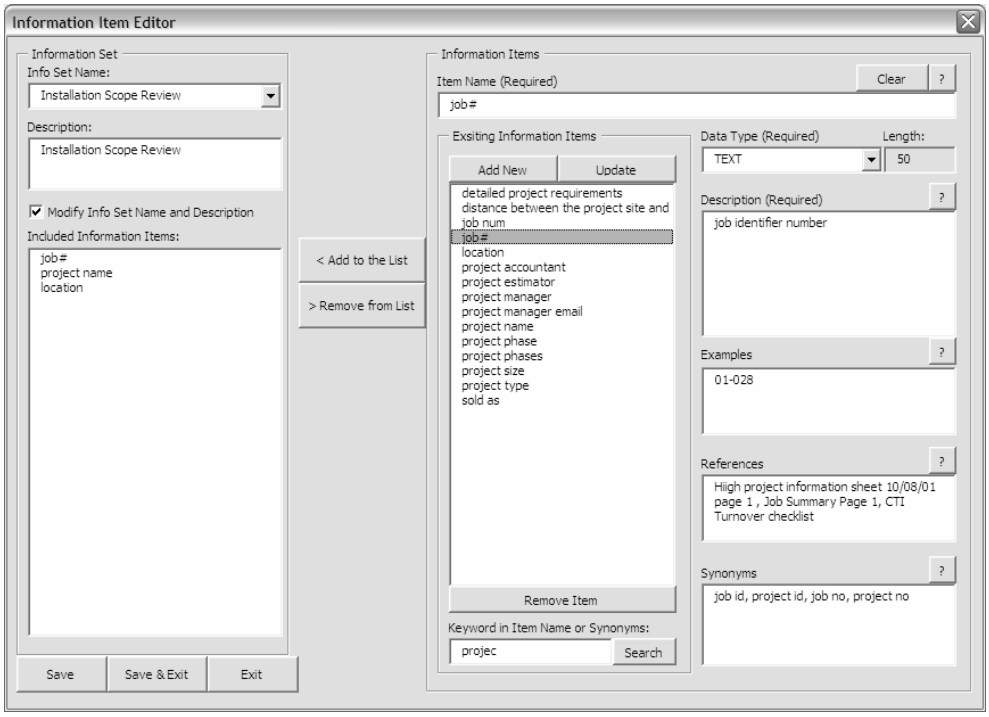


Figure 6.8 The Vernacular Information Item (VII) (or VDD) editor

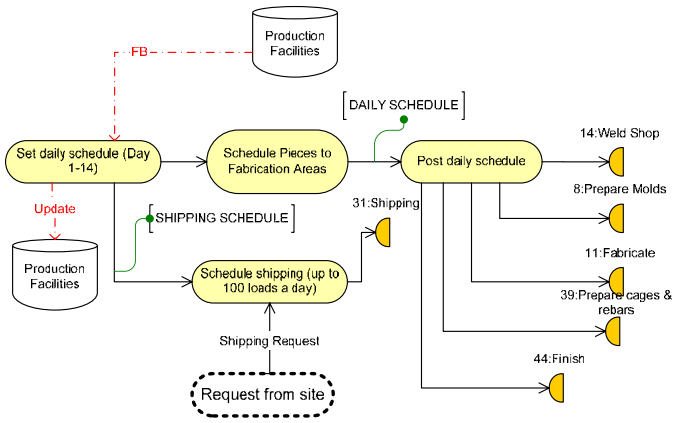


Figure 6.9. A part of a GT PPM model with information sets

Domain experts can define and add information sets and subsumed VIIs where they are necessary prior to defining information items required by each activity. DAILY SCHEDULE and SHIPPING SCHEDULE in Figure 6.11 are examples of information sets. Specific descriptions of new VIIs must be added to a vernacular data dictionary (VDD).

Information sets can be specified using the Information Set Editor (Figure 6.10). Users can add, remove, and update information items of information sets. Tags, which show a list of information sets in a flow, automatically appear when information sets are defined. Once information sets are defined, they can be used over and over.

After adding information sets, domain experts fill in input and output information of each activity, checking the consistency of a model using information sets as targets for information generation using the Activity Information Editor (Figure 6.11). The Activity Information Editor lists input and output information of an activity (see Section 4.8 for details). Inconsistent information items appear highlighted in the unused-, unavailable- or the not-provided-information lists.

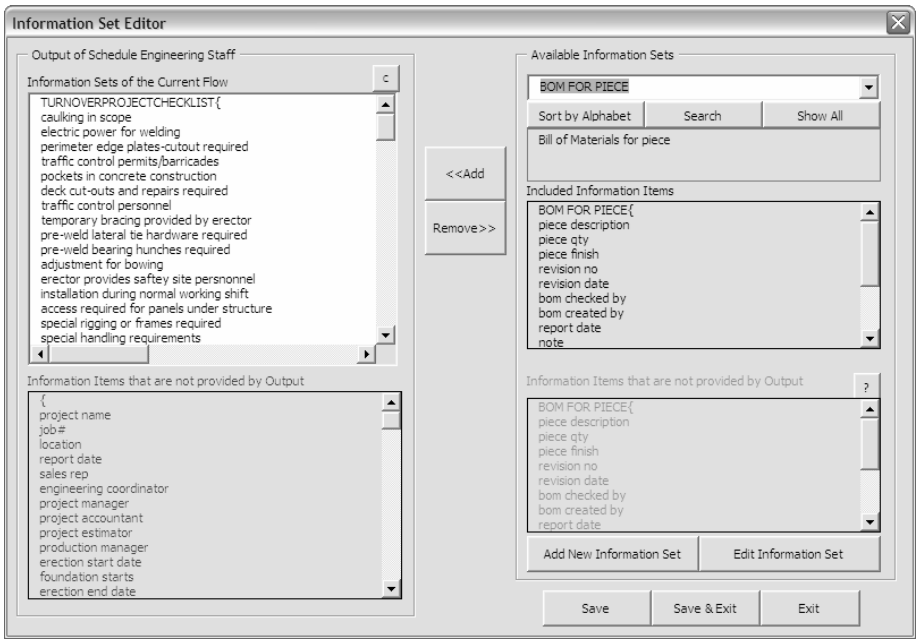


Figure 6.10 The Information Set Editor

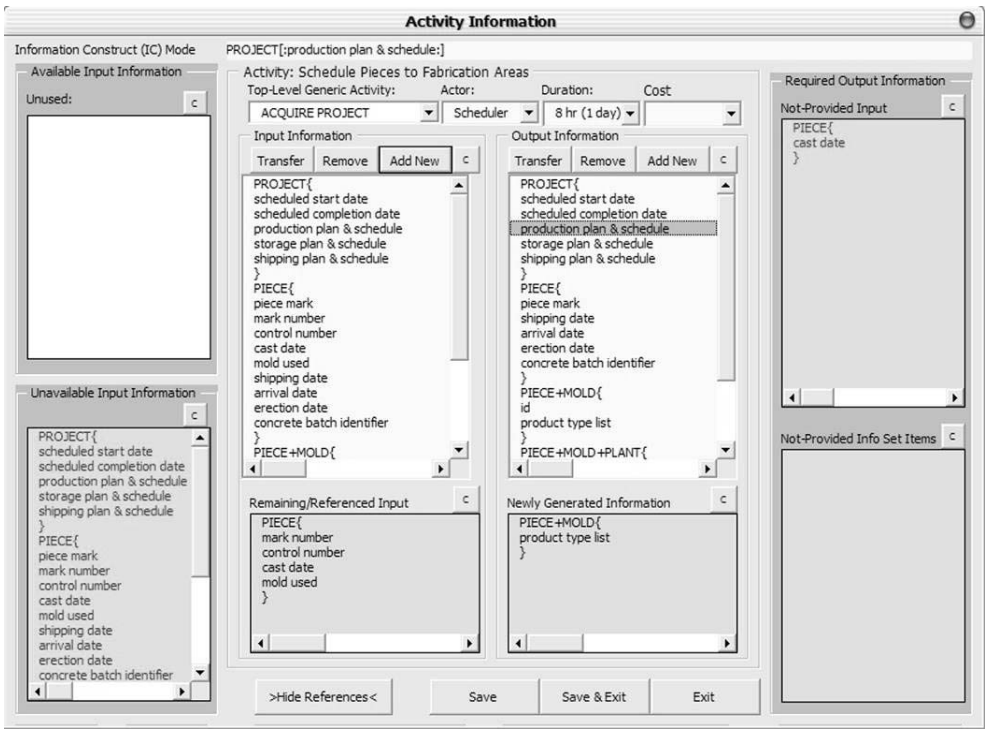


Figure 6.11. The GT PPM Activity Information Editor

If a project is simple in terms of the size of information and domain experts are comfortable with using information constructs (ICs), the VII modeling process can be skipped and information items can be specified using ICs from the beginning. However, if information items are defined in VIIs, the VIIs should be mapped to corresponding ICs using the Information Item Mapper illustrated in Figure 6.12. Currently one VII can be mapped to one or to many ICs. Sometimes one vernacular information item includes several pieces of information. But, in a non-computerized format, the subsidiary information items are not explicitly defined as individual information items. In such cases, the VII with several pieces of information can be mapped to several ICs. Or the VII can be decomposed into several VIIs. And each VII can be mapped to one IC. Conversely several VIIs should be mapped to one IC when VIIs are synonyms.

The structure and the contents of an information menu should be revised if ICs, which are meant to correspond with VIIs, cannot be composed from the information menu. The upper right corner window of the Information Item Mapper (Figure 6.12) shows mapped pairs of information items. The mapped VIIs can be automatically replaced by ICs. If there are any VIIs that are not mapped to ICs, a system automatically checks and lists them as 'unmapped information items' in the replacement procedure.

The Activity Information Editor (Figure 6.11) has the VII mode and the IC mode. Domain experts can switch freely from one information item mode to another. The consistency checking module works in both the VII mode and the IC mode. Domain experts fill in missing information or revise a model using the three methods described in Section 4.8.5 until a model becomes consistent.

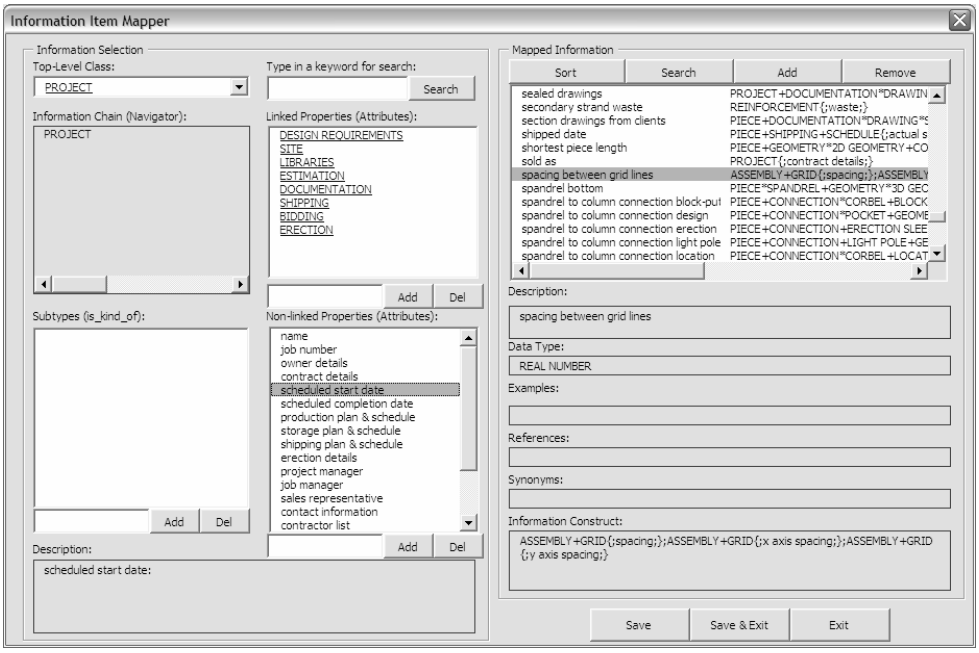


Figure 6.12. The GT PPM Information Mapper

GT PPM will automatically translate information items from one mode to another based on the mapping relations defined in the Information Item Mapper (Figure 6.12). The mapping rules are:

- From VIIs to ICs: If there are any newly defined VIIs, they will be translated as unmapped user-defined items. The unmapped VIIs can be mapped to ICs at this stage or at the next stage using the Information Mapper (Figure 6.12).
- Mediators collect GT PPM models from each domain expert. If there are still any unmapped VIIs, they should be mapped ICs at this stage by mediators in cooperation with domain experts.
- From ICs to VIIs: If there are no corresponding VIIs to ICs in a mapping table (Figure 6.12), the VIIs will be automatically named using corresponding ICs. Users can modify the VII names later.

6.3 THE LOGICAL PRODUCT MODELING (LPM) PROCESS

When the RCM process is completed, mediators extract ICs from collected RCM models. GT PPM automatically exports ICs of each activity to a new Excel® file (Figure 6.13). The extracted data can be further analyzed for various analyses.

	A	B	C	D	E	F
1						
2	File Name	Page Name	Activity ID	Activity Name	Actor	Information Item
3	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{scheduled start date}
4	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{Scheduled Completion Date}
5	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{production plan & schedule}
6	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{storage plan & schedule}
7	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{shipping plan & schedule}
8	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{Piece Mark}
9	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{mark number}
10	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{control number}
11	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{Cast Date}
12	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{mold used}
13	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{shipping date}
14	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{arrival date}
15	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{erection date}
16	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE{concrete batch identifier}
17	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE+MOLD{ID}
18	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE+MOLD+PLANT{id}
19	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE+MOLD+PLANT{capacity}
20	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE+MOLD+PLANT{layout}
21	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE+MOLD+PLANT{work rate}
22	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PIECE+MOLD+PLANT{piece list}
23	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{scheduled start date}
24	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{Scheduled Completion Date}
25	GTPPM a2v8r1.vsd	Scheduling	Internal Detail	Schedule Pieces to Fabrication Areas	Scheduler	PROJECT{production plan & schedule}

Figure 6.13. Exported Information Items

Even though the targeted data modeling language is EXPRESS, since many commercial database management systems (DBMS) are relational database management systems, the GTPPM tool also supports automated SQL code generation. SQL code can be generated by first creating EXPRESS code and then converting the EXPRRESS code to an SQL code using the GT EXPRESS2SQL (Figure 6.15) built on top of the CIS2SQL® schema converter. The CIS2SQL® schema converter is developed by Seok-Joon You at Georgia Tech (You, Yang, and Eastman 2004).

In EXPRESS, the specialization relation can be either ONEOF or ANDOR. Currently GTPPM is not allowing the ANDOR relation in order to reduce the complexity of a model. Each number on the command button in the EXPRESS Code Generator corresponds to each step of LPM (Figure 6.14).

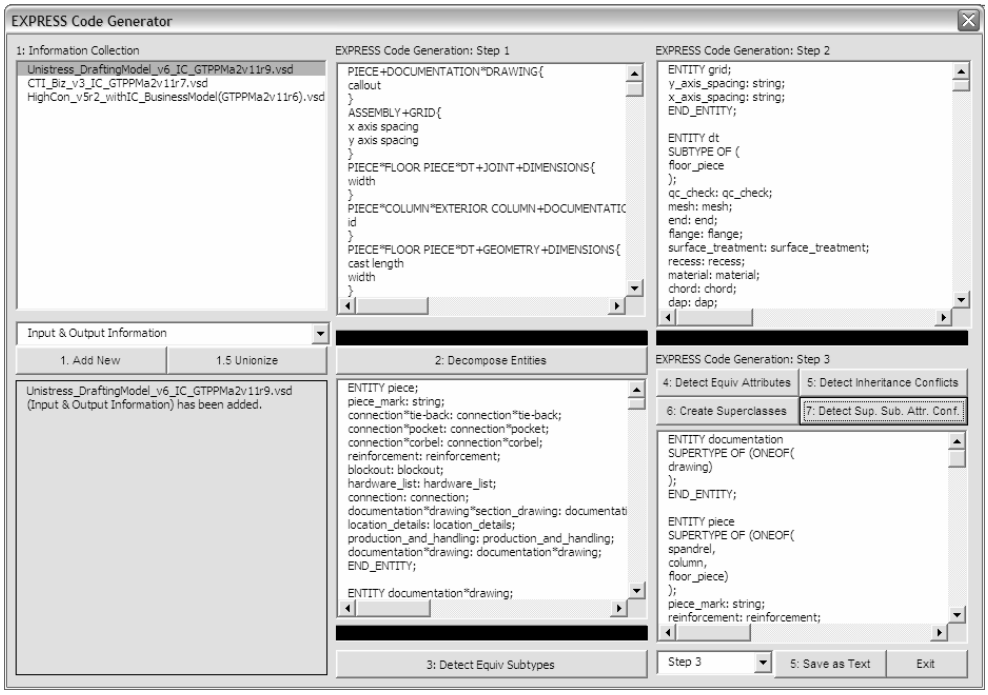


Figure 6.14 The EXPRESS Code Generator

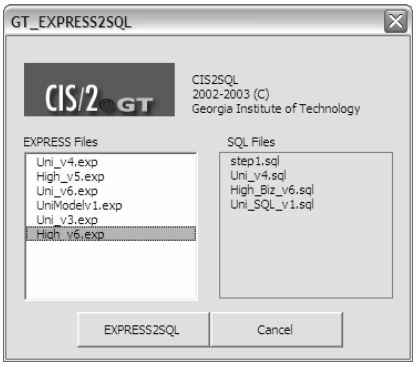


Figure 6.15 GT EXPRESS2SQL

CHAPTER 7

APPLICATION & EVALUATION

7.1 OVERVIEW

This chapter reports on the results of application and evaluation of GTPPM. The GTPPM has been deployed in the Precast Concrete Software Consortium (PCSC) project for several times for the last three years, and modified based on the results. The PCSC is a consortium of major precast concrete producers in Canada and the US[22] formed in 2001. The goals are to fully automate and integrate engineering, production, and construction operations, to gain productivity, and ultimately to increase the market share. As the means to achieve the goals, the PCSC chose to develop an intelligent 3D parametric CAD system and a Precast Concrete Product Model (PCPM) to enable data exchange between diverse systems used during the sales, design, engineering, production, and construction operations processes.

The following sections describe several GTPPM efforts. The PCSC member companies modeled their own management and engineering processes using GTPPM. As a

[22] Initially ITISA, a Mexican precast producer, was also a member of the PCSC. However, some of members have withdrawn and new members have joined the PCSC. The initial 23 member companies were Blakeslee Prestress, Cheyenne Concrete Co., Concrete Impression of Florida, Inc., Concrete Technology Inc., Con-Force Structures Ltd., Coreslab International Inc., Finfrock, High Concrete Structures, ITISA, IPC Inc., Lafarge Canada Inc., Meridian Precast & Granite, Metromont Prestress Company, New Enterprise Stone & Lime Co, Inc., Oldcaste Precast Inc., Pre-Con Inc., Rinker Precast, Rocky Mountain Prestress, Strescon Ltd., the Shockey Precast Group, the Spancrete Group Inc., Unistress Corp., and Wells Concrete Products Company. The current 15 member companies (as of March 29, 2004) are Blakeslee Prestress, Concrete Technology Inc., Con-Force Structures Ltd., Coreslab International Inc., High Concrete Structures Inc., IPC Inc., Lafarge Canada/Precon, Metromont Prestress Company, New Enterprise Stone & Lime Co. Inc., the Shockey Precast Group, Strescon Ltd., Tindall Corp., Unistress Corp., and Wells Concrete Products Company. The Georgia Tech team led by Prof. Charles Eastman and consisting of Rafael Sacks and Ghang Lee are technical advisors of the PCSC.

result, fourteen GTPPM models were developed. Among the fourteen GTPPM models, three models were elaborated based on on-site interviews. Information constructs collected from the three elaborated models were integrated and normalized into a single integrated product model. The integrated product model was compared to the PCC-IFC model, the IFC model extension for precast concrete (Karstila et al. 2002; Karstila and Suikka 2001; VTT 2004).

7.2 PROCESS MODEL PERSPECTIVES ON MANAGEMENT AND ENGINEERING PROCEDURES[23]

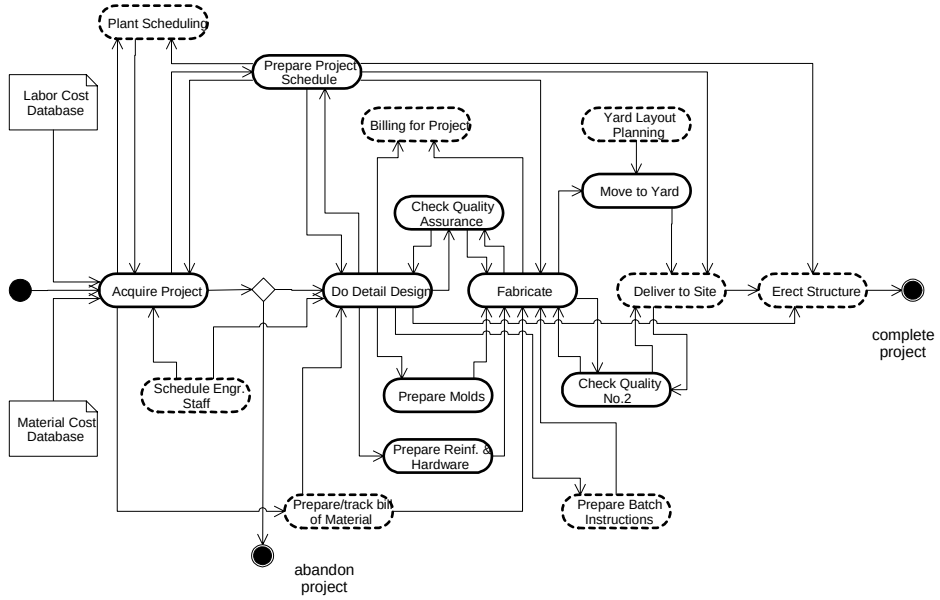


Figure 7.1 Generic top-level process model

From June 2001 to November 2001, GTPPM was deployed by fourteen PCSC members in analyzing the sales, design, engineering, and production processes of the precast concrete industry five years in the future. The goal was to understand and capture requirements for a next-generation precast concrete CAD system. The results were

[23] This section is a summary/excerpt from (Sacks, Eastman, and Lee 2004) with modification.

incorporated into a Request for Proposal (RFP) to CAD vendors. Typical processes began with a standard contract bid followed by the full range of precast concrete activities: cost estimating, bidding, contract award, assembly layout design, structural analysis, detailed piece design, production, handling, shipping, erection, scheduling and project control. The modelers' view was that of precast designers and producers, which defines the scope of the models. Client activities such as conceptual programming, overall project costing, and life cycle issues such as design for demolition and recycling, do not appear in any of them.

The collected models were categorized into three types: design build models, subcontract models, and design only models. Three models described a design-build process, and so covered the conceptual design phase in greater detail than the more traditional bidding process models. Two models were prepared by precast design consultants and so cover the design phase alone. Each model underwent a number of cycles of review by the research team and improvement by their authors before being approved for inclusion in the analysis and further development work. One model was rejected due to lack of detail, leaving thirteen models to work with.

All of the models use the generic top-level model as their starting point. Although modelers added additional intermediate layers of aggregate activities, every detailed activity can be traced to one common top-level activity. Using this as a starting point for analysis across companies, a list of middle-level activity groups was compiled for each top-level activity.

The degree of information dependence between activities was determined by the ratio of the number of information flow (n_F) to the number of detailed activities (n_A). Table 7.1 shows the degree of information dependence between activities by three model types.

The analysis results indicated that the degree of dependence between activities was relatively unvarying by model type. But, since the number of samples was small, we were reluctant to generalize the finding.

Table 7.1 The degree of information dependence between activities by model type

Model Type (1)	**Feature*** (2)	**Average** (3)	**Largest** (4)
Design Build Models	n_A	269	323
	n_F	476	572
	n_F / n_A	1.77	1.77
Subcontract Models	n_A	154	275
	n_F	232	520
	n_F / n_A	1.50	1.89
Design Only	n_A	57	81
	n_F	89	130
	n_F / n_A	1.56	1.60

*: n_A = number of activities; and n_F = number of information flows

While analyzing the collected information constructs, inconsistency in information flows was found. This motivated the development of a more rigorous method to validate the consistency of information flow as described in Section 4.8.2.

7.3 PRODUCT MODELS FOR MANAGING ESTIMATION, SCHEDULING, AND SHIPPING INFORMATION

In December 2002, GTPPM was deployed for the second time in a project to capture the current management processes (i.e., estimation, bidding, production, and shipping) of two precast producers, High Concrete (Denver, PA) and CTI (Springboro, OH) after major modification. Unlike the first attempt, the models were generated by the author based on the interviews with the manager-level personnel of each company (Figure 7.2).

Figure 7.2 A round table discussion at High Concrete before one-on-one interviews

Later, the generated models were reviewed again by domain experts. The two companies were chosen because they were two of a few companies, which had a database management system for managing estimation, production, and shipping information. The goals were to capture their current processes and information flow as they were, and to compare automatically generated (preliminary) data models and their actual database schemas.

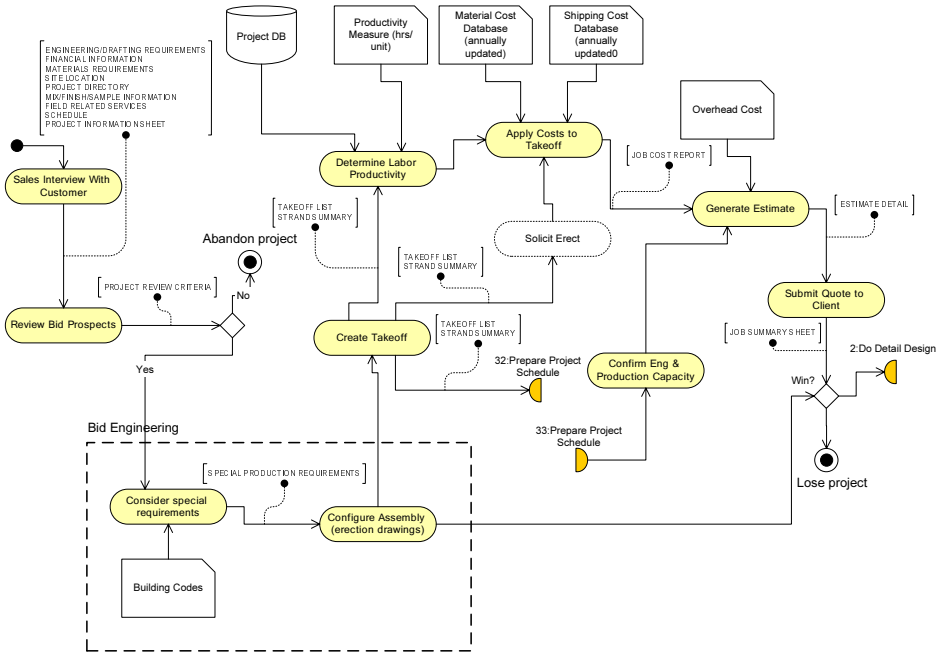


Figure 7.3 Acquire Project

First, process models were generated with domain experts at each department. Figure 7.3 illustrates a process of “Acquire Project” with information sets required by a project acquisition process. During this process, takeoff (i.e., the quantity of products and subcomponents), rough estimation and production schedule, and bidding information were generated.

Then, information sets were defined based on standard company reports required by the end of certain activities (e.g., job summary sheet, turnover meeting check list, piece tag, and packet slip). The information sets were defined with vernacular information items (VIIs). Examples of specified information sets and their items are as follows:

```
PROJECT INFORMATION SHEET {;project name;location;report
date;purchaser;address;city_state_zip;project size;job#;contract
value;taxes;status;type;sold as;detailed project requirements;Sales Rep;estimator;}

PIECE DRAWING {;piece mark;piece qty;piece volume;piece weight;hardware/reinforcing
item;hardware/reinforcing quantity;mix #;revision date;revision by;revision
no;drawn date;drawn by;dwg ckd;eng chk;dwg ckd date;eng ckd date;project
name;drawing nbr;job#;dimension;piece shape;material pattern;note;received
date;issued date;concrete strength;dwg destroy date;rebar schedule;}

PACKING SLIP {;address;city_state_zip;job#;truck number;trailer number;truck
driver;payment method;po#;piece mark;piece qty;piece description;comments;contents
packaged by;contents checked by;contents received by;delivered date;}

PIECE TAG {;bar code;piece weight;piece mark;}

BOM FOR PIECE {;project name;job#;project phase;piece mark;drawing nbr;note;report
date;bom created by;bom checked by;revision date;revision no;piece finish;piece
qty;piece description;}

SCHEDULE{;Contract Date;engieering date;review by architect;production end
date;erection start date;erection end date;}
```

JOB COST REPORT {;project name;location;project type;job#;estimate no;product type id;product element id;operation;product size;product u/m;product qty;operation cost;total operation cost;}

TAKEOFF LIST {;project name;location;job#;product type id;product element id;product name;product qty;product size;product u/m;estimator;estimate no;area code;distance between the project site and the plant;piece mark;piece depth;piece width;piece unit length;piece weight;load name;total loads;total # of pieces;piece qty;}

JOB SUMMARY SHEET {;project name;location;estimate no;rev no;job#;product name;product type id;product qty;product size;product u/m;product $/unit;product amount;total production cost;total yard costs;total shipping cost;total erection cost;taxes;total markup;gross margin without markup;gross margin;total bid price;scope of work;}

Some other examples of information sets without detailed items include:

PROJECT DIRECTORY

SITE LOCATION

FINANCIAL INFORMATION

ENGINEERING/DRAFTING REQUIREMENTS

MATERIALS REQUIREMENTS

MIX/FINISH/SAMPLE INFORMATION

FIELD RELATED SERVICES

BOM FOR HARDWARE

PROJECT REVIEW CRITERIA

ESTIMATE DETAIL

ERECTION DRAWING

STRAND SUMMARY

SPECIAL PRODUCTION REQUIREMENTS

DETAILED POUR SCHEDULE

DAILY CONCRETE POUR SCHEDULE

TURNOVER MEETING CHECK LIST

```
DAILY PRODUCTION SCHEDULE
4 WEEK SCHEDULE
PRODUCTION SCHEDULE
FORM DRAWING SCHEDULE
PRE-TENSION REPORT
```

The specified VIIs were mapped to ICs using the Information Item Mapper (Figure 7.4). VIIs and ICs were generally mapped one to one. However, several VIIs and ICs were mapped many to many. Some VIIs, which were synonyms, were mapped to an IC. Some VIIs, which were defined as one information item, but actually included several pieces of information, were mapped to several ICs. An example of the latter is `galvanized embed order status`. In order to keep track of the order status of a product or a part in terms of a data management, we need to specifically know which item has been ordered, what is the purchase order identifier, and so on. However, when such information is maintained in a paper format, it is recorded informally and freely as one long note. Based on the data recorded in `galvanized embed order status`, the `galvanized embed order status` was mapped to several ICs as follows:

```
PIECE+MATERIAL*HARDWARE{;type;};
PIECE+MATERIAL*HARDWARE{;id;};
PIECE+MATERIAL*HARDWARE+PURCHASE_ORDER{;status;};
PIECE+MATERIAL*HARDWARE+PURCHASE_ORDER{;id;}
```

Some VIIs had a different meaning than what they seemed to mean. A VII `rebar schedule` is a good example. `rebar schedule` is not a type of regular time-based schedule, but is a common term in AEC that denotes a 2D abstract representation of bent rebar. In the mapping process, some of ambiguous VIIs such as `rebar schedule` were mapped to ICs

based on the definitions, data types, examples, references, and synonyms of the VIIs (the right side of Figure 7.4).

The specified VIIs in information sets were automatically converted to ICs according to the mapped relations between VIIs and ICs. Input and output information of activities were specified using information sets as a target of information production. The consistency of information flow was checked. As a result of these two modeling processes, 135 and 231 distinctive information constructs were collected respectively from the High and the CTI models.

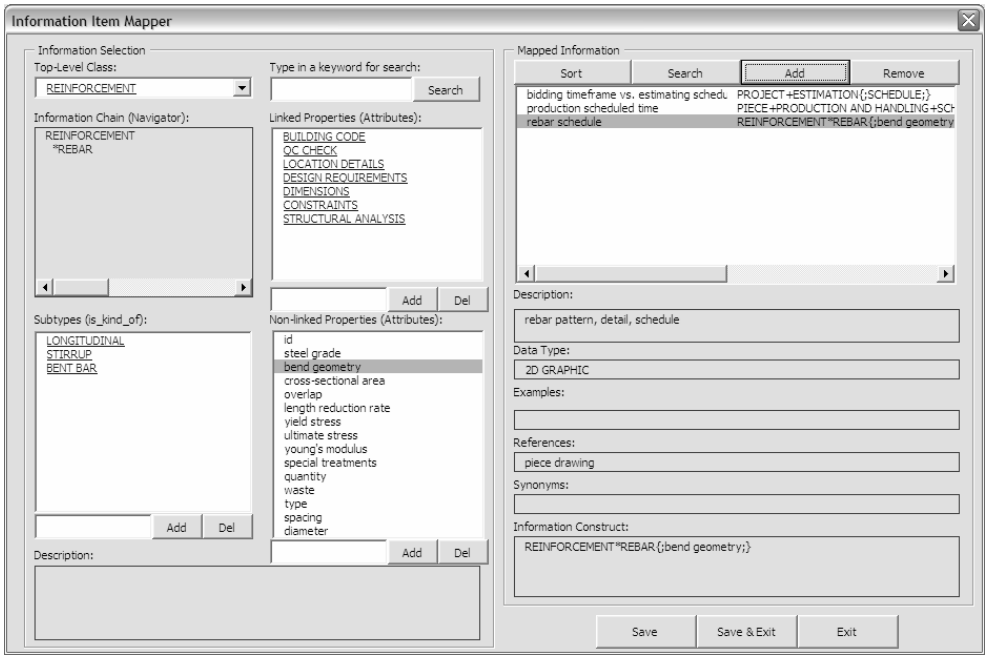


Figure 7.4 Mapping ambiguous terms based on the descriptions

In the beginning, there were some concerns about the possibility of the GTPPM modeling process being too tedious and time-consuming because it requires very detailed process and information flow modeling. It was important to measure the modeling hours

because GTPPM would not be an appropriate substitute for the current modeling method and process if it takes relatively too much time.

Table 7.2 The statistics of the High model

	Modeling Hours	**Statistics**
Process modeling	3 days (24 hours) 11/25-27, 2002	Internal Detail: 98 External Detail: 13 Internal Highlevel: 9 External Highlevel: 13 Information Flow: 210 Feedback Flow: 14 Material Flow: 70 Dynamic Repository: 10 Static Information Source: 6 Continue: 84
VIIs modeling	12.5 hours	Information Sets: 24 VDDs: 192 (non-distinctive)
Mapping VIIs to ICs, Revision of an IM	7.5 hours	ICs: 135 (distinctive)
Total	**44 hours**	

Table 7.3 The statistics of the CTI model

	Modeling Hours	**Statistics**
Process modeling	3 days (24 hours) 12/18-20, 2002	Internal Detail: 96 External Detail: 29 Internal Highlevel: 7 External Highlevel: 29 Information Flow: 179 Feedback Flow: 9 Material Flow: 64 Dynamic Repository: 14 Static Information Source: 10 Continue: 42
VIIs modeling	3 hours	Information Sets: 6 VDDs: 186 (non-distinctive)
Mapping VIIs to ICs	2 hours	ICs: 231 (distinctive)
Total	**29 hours**	

The modeling hours for the High and CTI models were recorded. Table 7.2 and Table 7.3 show statistical data of the High and the CTI models. The whole RCM modeling process took about 37 hours in average. 97 internal detail activities, 195 information flows, and 15 information sets were defined in average. There was no significant difference

between two models in terms of the number of process components or the number of information constructs. 37-hour work is about 5-day (a week) work. It seemed pretty reasonable if one could develop a product model within a week or even a month considering some preparation and revision time before and after GTPPM modeling.

The automatically collected High's and CTI's information constructs were normalized into two separate preliminary product models in EXPRESS. In order to compare the results with the data structures of High's current database management system, the information constructs collected from High's model were also normalized into a SQL schema. In this process, a new SQL generation module was developed and used to show referential relationships between TABLEs because the EXPRESS2SQL module does not generate referential relations between TABLEs.

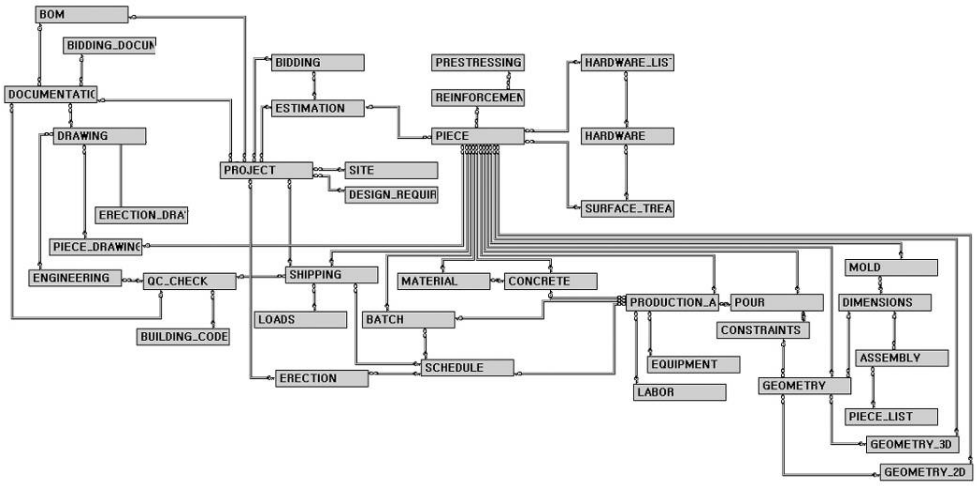


Figure 7.5 A SQL table structure of the High model with referential relations

Figure 7.5 graphically shows a SQL table structure of the High model with referential relations. This diagram was sent to the information system (IS) manager of

High with SQL code for review. The author visited High for the second time to interview High's IS manager.

Currently High's ERP system is a federated database management system, which is composed of several commercial and custom-built database management systems. High was using an MS Access®-based estimation system, two Oracle®-based production scheduling, shipping, inventory, purchase management systems, a legacy accounting/costing system, an engineering/drawing management system, and a human resource/payroll system. However, only limited sets of information can be exchanged between different database management systems today. Currently High is developing a central database that can integrate the dispersed databases and also that can acquire geometric information and bills of materials (BOMs) directly from an advanced 3D CAD system.

A one-to-one comparison between the automatically generated data model and High's data schemas was not possible for several reasons. First, the automatically generated model was designed as one large schema, but High's system was a federated databases. Second, the automatically generated data model was based on an object-oriented modeling approach (i.e., EXPRESS) whereas High's systems were relational databases using SQL. Conceptually SQL TABLEs are correspondent to Entities in EXPRESS. However, because of lack of the inheritance mechanism in relational database and several practical implementational reasons, data modelers in field (i.e., IT managers) tend to put as many number of attributes in one TABLE as possible rather than to break down an entity into an *atomic* level (i.e., a semantically indecomposable level). It is to achieve the efficiency in table management and also to reduce the complexity of the JOIN operation in

query. Third, the terms used to define TABLEs and attributes in High's systems were different from those used to define entities and attributes in the automatically generated schema. Thus, it was very difficult to automatically or quantitatively compare the two schemas. The evaluation had to rely on qualitative and subjective evaluation of the author and the IS manager at High.

The automatically generated SQL model included thirty-nine TABLEs. Each TABLE and its attributes were reviewed. After reviewing the TABLEs, High's IS manager and the author categorized TABLEs into three groups:

1) *Over-defined*: TABLEs that include more information than High's current data models
2) *Adequate*: TABLEs that define information about the same level as the current High's data models
3) *Under-defined*: TABLEs that lack necessary information

In overall, the automatically generated product model included more information than what was maintained by the current database management systems. Currently only little geometry, shipping, loading, constraints, and engineering information is managed by database management systems. Also (concrete) mold information is not maintained because mold design varies project by project and they thought that it was unnecessary to keep track of mold information. The automatically generated model included quite a few "over-defined" information items because the initial process model was developed based on an assumption that High would adopt a new advanced 3D modeling system, which would be equipped with many automated engineering and constraint checking functions.

Table 7.4 Evaluation of the High Model

Over-defined	Adequate	Under-defined
ASSEMBLY	DESIGN REQUIREMENTS	BATCH (mix recipe)
BIDDING	DOCUMENTAION	CONCRETE (mix recipe)
BOM	DRAWING	
BUILDING_CODE	ERECTION_DRAWING	
CONSTRAINTS	ESTIMATION	
DIMENSTIONS	HARDWARE	
ENGINEERING	HARDWARE_LIST	
EQUIPMENT	LABOR	
ERECTION	MATERIAL	
GEOMETRY (2D, 3D)	PIECE	
MOLD	PIECE_DRAWING	
QC_CHECK	PIECE_LIST	
SHIPPING	PRODUCTION_AND_HANDLING	
SURFACE_TREATMENT	POUR	
TRUCK_LOADS	PRESTRESSING	
	PROJECT	
	REINFORCEMENT	
	SCHEDULE	
	SITE	

On the other hand, the automatically generated model lacked the batch and concrete information, especially the concrete mix design (a.k.a. “mix recipe”) information. In the actual ERP system, the concrete mix design information was managed through a couple of large TABLEs while the automatically generated model defined concrete mix information simply as `mix_specification`.

```
ENTITY concrete
    SUBTYPE OF (
    material
    );
    mix_specification: string;
    strength: string;
END_ENTITY;
```

It is because the mix design information is inputted directly from the field (i.e., a batch plant) and the domain expert and the author, who modeled the process and

information flow, had not had a chance to interview anybody related to the concrete mix design. As a result, the concrete mix design information was only captured as a simplistic form. This reconfirmed the fact that GTPPM can derive a product model only from the specified scope and information requirements.

High's IS manager evaluated that the automatically generated product model generally reflected High's information requirements well. According to the comparison results, the RCM models and the LPM process have been modified. Currently GTPPM can selectively collect information items that are actually stored and managed by a database management system by using the Dynamic Repository shape.

7.4 PRODUCT MODELS FOR DESIGNING/DRAFTING

GTPPM was deployed for the third time to capture a precast concrete "designing/drafting" process. Engineering and designing/drafting processes are not easy to capture because of the domain expertise included in them and also because of the complexity of the processes. Even for domain experts with more than 10 years of experience, it is still not easy to describe engineering and modeling processes in a systematic way unless they sit down and spend some time on thinking about them. Fortunately, Unistress, a precast producer in Pittsfield, MA, provided detailed guidelines for designing precast concrete pieces. Based on the guidelines, the designing/drafting processes for double tees (Figure 7.6) and exterior columns were modeled.

Unlike the previous modeling processes, information items of each activity were directly defined without using information sets. They were first defined as vernacular information items (VIIs) and then mapped to information constructs (ICs) later.

Figure 7.6 A stack of double tees

The major difference between a business management process and a designing/drafting process in terms of information flow is that information flow in the designing/drafting process is accumulative: i.e., a model of a precast concrete structure behaves as a data repository. As soon as a designer adds one shape or texts to a precast concrete model or to a drawing, they represent certain information. But such design information does not only affect only the next activities, but also many other activities that appear later in the process. Thus, a model of a precast concrete piece in this case study was represented as a dynamic repository as shown in Figure 7.7.

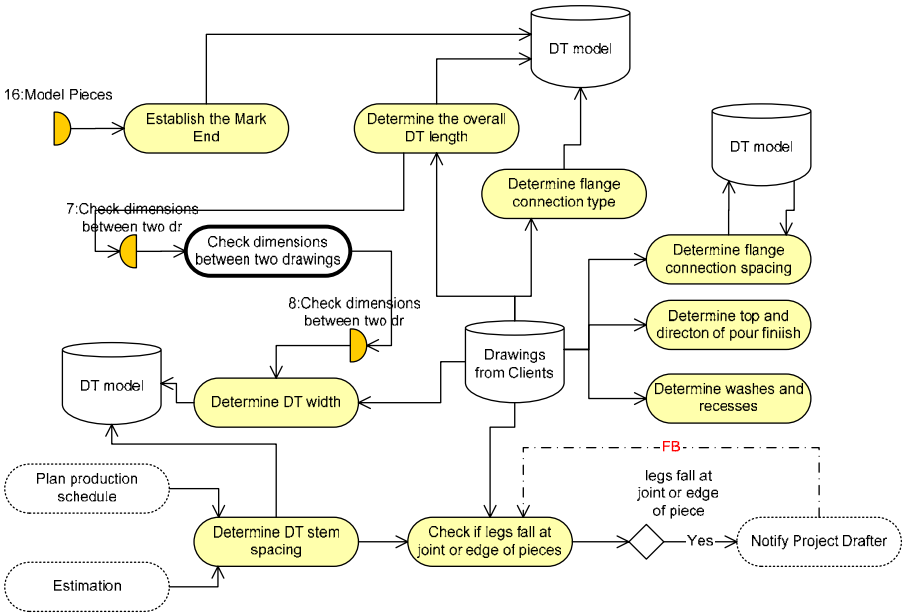


Figure 7.7 A part of a double tee modeling process

Even though “Drawings from Clients” cannot be changed by precast concrete designers, they are also represented as a dynamic repository in Figure 7.7 because they can updated by clients many times during a project. Figure 7.8 illustrates a process of receiving drawings from clients.

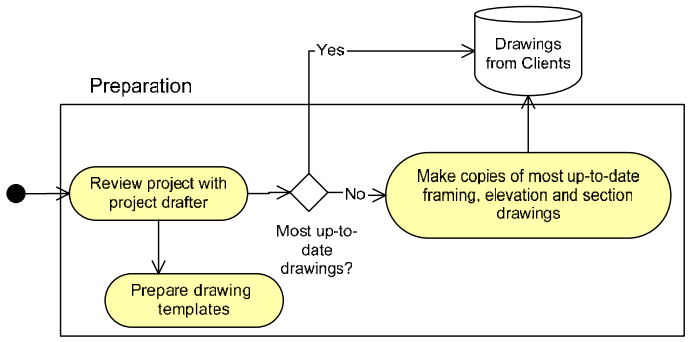


Figure 7.8 Drawings from clients

Table 7.5 The difference in the PIECE definitions

The High model	The Unistress model
ENTITY piece; estimation: estimation; piece_drawing: piece_drawing; material: material; mold: mold; reinforcement: reinforcement; geometry: geometry; piece_mark: string; product_unit_measurement: string; product_size: string; product_amount: string; product_name: string; product_code: string; label: string; surface_treatment: hardware_list: hardware_list; production_and_handling: shipping: shipping; END_ENTITY;	ENTITY piece SUPERTYPE OF (ONEOF(spandrel, pc_column, floor_piece)); piece_mark: string; reinforcement: reinforcement; blockout: blockout; hardware_list: hardware_list; connection: connection; location_details: location_details; production_and_handling: drawing: drawing; END_ENTITY;

Another difference between the previous High and CTI models and the Unistress model is that the Unistress model includes specific types of products. For example, Table 7.5 shows the definitions of the `piece` entity, a main product of the precast concrete

industry, in the High and the Unistress models. Since the High model focuses on the management process, types of `pieces` are defined by generic information such as `product name` or `piece_mark` whereas, in the Unistress designing/drafting model, types of `pieces` are defined specifically as `spandrel`, `pc_column`, or as `floor_piece`. It is because, in order to design a piece, designers need to know specifically which type of piece is connected to which type of piece. By the same reason, even though we only focused on the processes of designing/drafting double tees, the definitions of adjacent pieces and connections, whose information is required to design a double tee, were also captured in the derived product model. (See Figure 7.11 in the next section for an EXPRESS-G diagram of the expanded `piece` and `connection` definitions.)

Table 7.6 The statistics of the Unistress model

	Modeling Hours	**Statistics**
Process and VIIs modeling for a double tee modeling process	6 hours	Internal Detail: 55 External Detail: 7 Internal Highlevel: 4 External Highlevel: 7 Information Flow: 160 Feedback Flow: 3 Material Flow: 0 Dynamic Repository: 21 Static Information Source: 2 Continue: 18
Process and VIIs modeling for a column modeling process	2 hours	Information Set: 0
Mapping VIIs to ICs	2 hours	IC: 85 (distinctive)
Total	**10 hours**	

The Unistress model was about half size of previous models in terms of both the number of process components and the number of distinctive information items because it only dealt with a small portion of the design and engineering process. It took 10 hours to

model the Unistress model. The Unistress model included 73 activities, 163 flows, and 85 information constructs. The automatically generated product model included 58 entities.

7.5 THE INTEGRATION AND EVALUATION OF AUTOMATICALLY GENERATED PRODUCT MODELS

Information constructs collected from three models were integrated as one model through the LPM process. The integrated model included 129 entities and modeling of the three companies' processes took 73 hours in total. For readers' reference, CIS/2 LPM 6 has 731 entities and PCC-IFC Version 0.9 has 413 entities. The automatically generated integrated models are provided in Appendix G.

The syntax of automatically generated integrated product models has been validated using the syntax checkers embedded in a commercial tool EXPRESS Data Management (EDM®) Supervisor Version 4.5 (Figure 7.9) and a shareware Expresso Version 3.1.4. The automatically generated schemas could be successfully implemented as physical data models both on MS SQL Server 2000® and EDM® as they were without further refinement and modification.

In the integrated model, we could observe several problems. Figure 7.10 is a hierarchy (called, an *entity graph* in Expresso) of `MATERIAL` generated by the Expresso Entity Grapher. The entity graph shows a specialization hierarchy of entities. Even though we were extremely careful to avoid the 'nym' issues, we can observe from Figure 7.10 that `reinforcement` in the model was used in two meaning: reinforcement as an activity and also as a material (object). It is because the information menu was initially defined violating the 'nym' principle. This problem was fixed later.

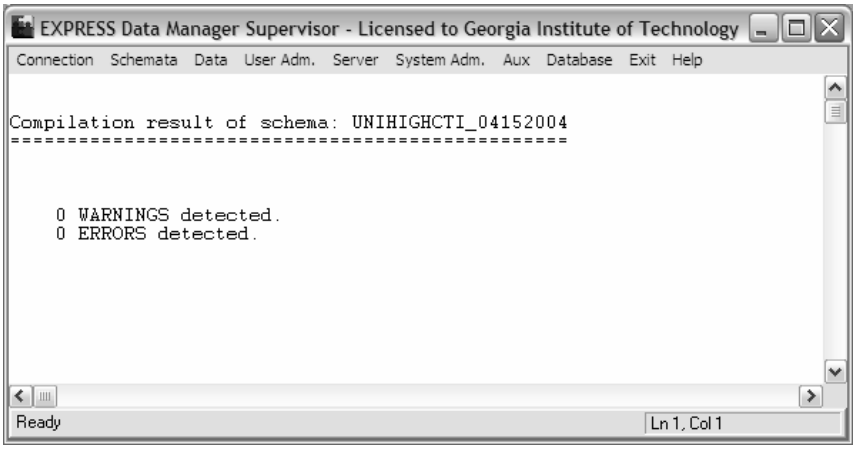


Figure 7.9 EXPRESS code validation by EDM®

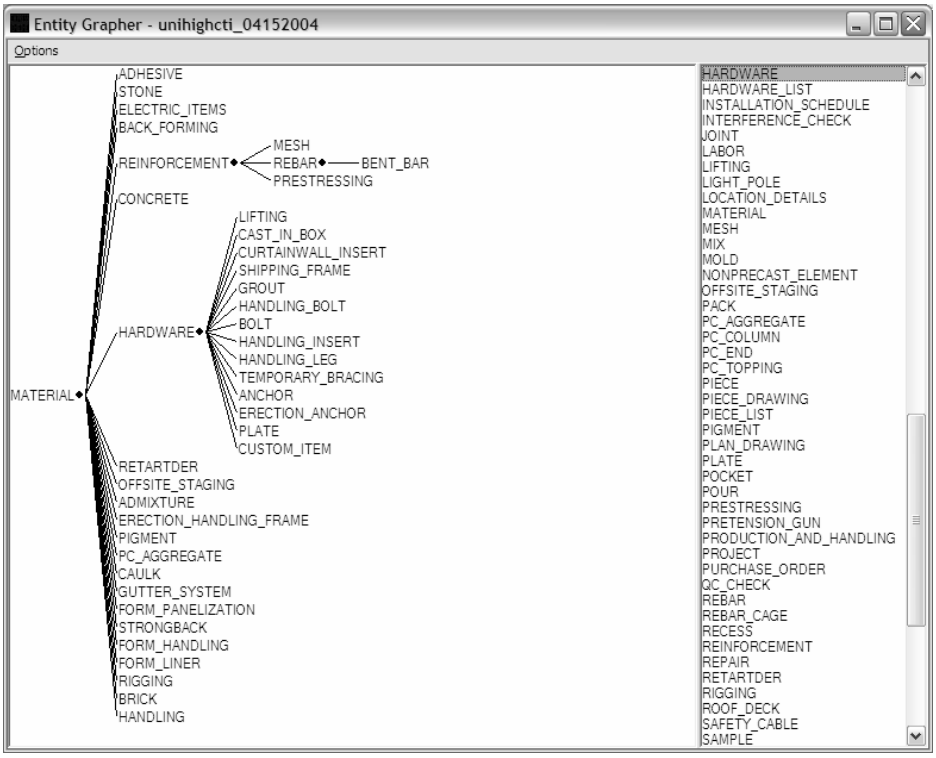


Figure 7.10 A hierarchy of MATERIAL generated by the Expresso Entity Grapher

On the other hand, the level of detail of the automatically derived model is generally satisfactory. The model defined information at the level of detail that is required for the targeted purposes: i.e., managing and designing pieces. Figure 7.11 illustrates an

EXPRESS-G model of the integrated piece and connection definitions from the High, CTI, and Unistress models. `dt` in the model represents the double tee entity. The direct association relations between `dt` and two connection types `dap` and `chord` in Figure 7.11 can be refined by the WHERE clauses in the manual modification process.

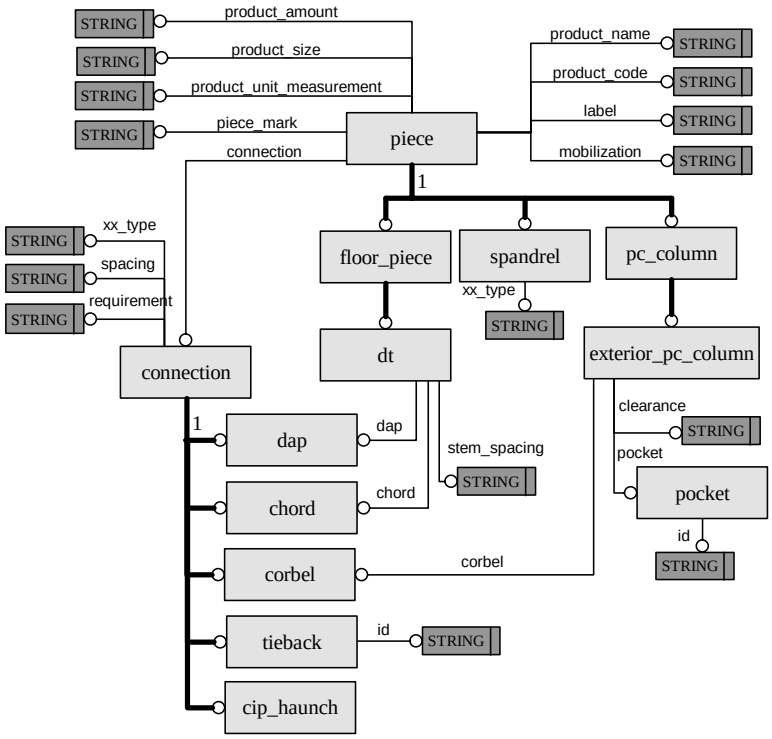


Figure 7.11 Automatically generated `PIECE` and `CONNECTION` definitions

Figure 7.12 shows several other examples of entity hierarchies in the integrated model.

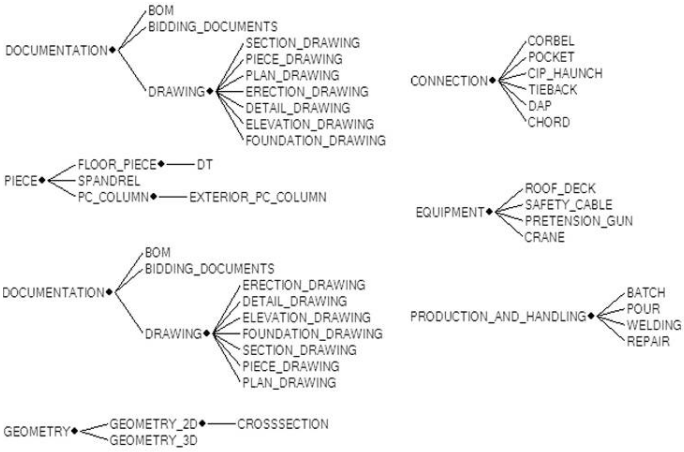


Figure 7.12 Several entity graphs of entities in the integrated model

A good benchmark of the integrated model might be the PCC-IFC model, a precast concrete extension to an existing IFC model. As described earlier, IFC models are built based on a conceptually modeling approach. As a result, they have a weak connection with real use cases and are defined at a relatively high level. For example, Figure 7.13 shows an entity graph of `IFC Building Elements`. The IFC entities that are corresponding to spandrels, columns, and double tees in the integrated model (Figure 7.11) are `ifcbeam, ifccolumn`, and `ifcslab` in (Figure 7.13).

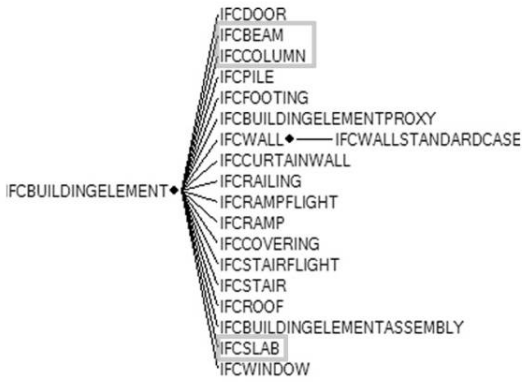


Figure 7.13 An entity graph of IFC Building Elements

Figure 7.14 is a partial EXPRESS-G model of these three IFC building elements. As shown in Figure 7.14 and the following EXPRESS code, the PCC-IFC model only defines the object names and do not have any attribute. It assumes that all the attributes will be inherited from supertypes.

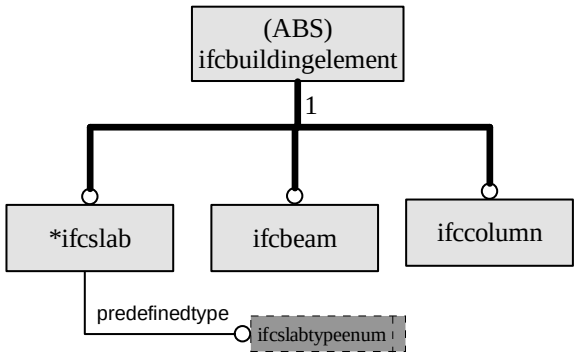


Figure 7.14 A partial EXPRES-G model of IFC Building Elements

```
ENTITY IfcBuildingElement
    ABSTRACT SUPERTYPE OF (ONEOF(
      IfcBuildingElementProxy
     ,IfcBeam
     ,IfcColumn
     ,IfcCovering
     ,IfcCurtainWall
     ,IfcDoor
     ,IfcRailing
     ,IfcRamp
     ,IfcRampFlight
     ,IfcRoof
     ,IfcSlab
     ,IfcStair
     ,IfcStairFlight
     ,IfcWall
```

```
     ,IfcWindow
-- Additional subtypes defined by ST-3
     ,IfcBuildingElementAssembly
     ,IfcFooting
     ,IfcPile
    ))
    SUBTYPE OF(IfcElement);
    INVERSE
      ProvidesBoundaries : SET OF IfcRelSpaceBoundary FOR RelatedBuildingElement;
      HasOpenings        : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
      FillsVoids         : SET [0:1] OF IfcRelFillsElement FOR
RelatedBuildingElement;
  END_ENTITY;

ENTITY IfcColumn
    SUBTYPE OF(IfcBuildingElement);
END_ENTITY;

ENTITY IfcBeam
    SUBTYPE OF(IfcBuildingElement);
END_ENTITY;

ENTITY IfcSlab
    SUBTYPE OF(IfcBuildingElement);
      PredefinedType : IfcSlabTypeEnum;
    WHERE
      WR2 : (PredefinedType <> IfcSlabTypeEnum.USERDEFINED) OR
                    ((PredefinedType = IfcSlabTypeEnum.USERDEFINED) AND
        EXISTS(SELF\IfcObject.ObjectType));
END_ENTITY;

TYPE IfcSlabTypeEnum = ENUMERATION OF
    (FLOOR,
     ROOF,
     LANDING,
     USERDEFINED,
```

```
      NOTDEFINED);
END_TYPE;
```

Since the IFC model is still growing, it will not be valid to argue the goodness or the badness of the model based on its level of details. And the intention of the comparison is not to judge the goodness of the model. This comparison shows the level of details that GTPPM can capture and the possibility of GTPPM to capture a more practical and realistic set of data, which is sensitive to its use cases.

CHAPTER 8

Conclusion

Product modeling is not art that depends only on intuition and subjectivity, but science that depends on logical thinking and explicit procedures with clear objectives that can be tested and improved upon. However, existing requirements collection methods of product modeling rely solely on human review and suffer from a logical gap between their Application Activity Model (AAM) and Application Requirement Model (ARM). The existing methods have more significant problems when applied to large and heterogeneous business environments. Any review process will get slower and collected information will get more difficult to check because the number of information items will grow large. There have been several research and development efforts to overcome these drawbacks, but none provides any formal method and procedure to elicit and validate information items of a domain and to (semi-)automatically derive a product model from collected information requirements.

The author proposed a formal Requirements Collection and Modeling method (RCM) and Logical Product Modeling (LPM). RCM enables modeling and domain experts to capture the contents, scope, granularity, and semantics of information used in the activities of a process. LPM provides the logic of integrating and normalizing information constructs collected from RCM models into a preliminary product model.

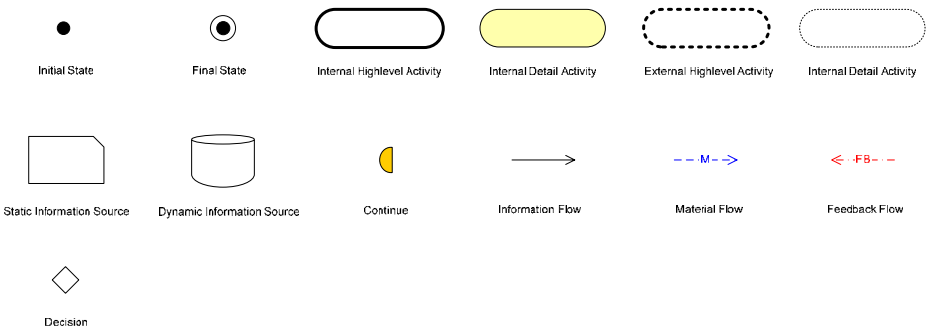


Figure 8.1. RCM Notation

The characteristics of RCM are that it 1) is *information-specific* so that it can capture the information items used in the activities making up the process; 2) guarantees the *completeness* of the product model data in relation to the process models defining the UoD; 3) provides rigorous syntax and checking methods that can help modelers maintain *consistency* (i.e., logical coherence) in their models; 4) allows modelers *to express heterogeneous business environments* how each company deploys and uses information in its business process. (The goal of the requirements collection method is to collect and integrate information items within an industry-wide product model. However, this does not necessarily require the definition of a unified process model); and 5) supports a *step-by-step* modeling procedure that can guide domain and modeling experts to elicit requirements and information and to transform them into a process and information-flow model in a step-by-step manner. More generally, by making the process explicit, the results from each step can be analyzed and criteria for success of each of the steps developed, allowing a science of process-to-product modeling to be developed.

By allowing modelers to specify information in a process (in the context of its use) step-by-step and providing a logical and dynamic consistency checking method, RCM helps modelers to capture complete and realistic information.

LPM defines nine design patterns to automatically integrate and normalize information constructs. It decomposes, generalizes, and restructures a set of information constructs into a preliminary product data model. We expect that the number of these design patterns will grow in the future similar to the normal forms in database.

However, the GTPPM method is by no means complete. An automatically generated product model will not include roles, data type, cardinality, and the WHERE, DERIVE, and RULE clauses. Those should be added and modified manually. In the future, the logic of further automating those processes can be provided. For example, it might be possible to define the DERIVE relations between attributes using the functional dependencies between input and output information defined in an RCM model.

GTPPM has been experimented with the precast concrete producers in the North America. Through the application and evaluation of GTPPM, several drawbacks as well as advantages are identified. GTPPM has been modified based on the findings. However, some of those were left as the topics of future work.

By using GTPPM, a complete set of information items required for product modeling for a medium or a large industry can be collected without generalizing each company's unique process into one unified high-level model. However, the use of GTPPM is not limited to product modeling. It can be deployed in several other areas including:

- workflow management system (Jablonski and Bussler 1996; P. Lawrence (Ed.) 1997; WFMC 1999) or MIS (Management Information System) development: Information required for processing an activity, passed to succeeding activities, and returned back to previous activities for feedback can be defined. (See Appendix H for details on workflow management systems.)

- software specification development: A detailed definition of engineering functions and processes can be developed, which will allow further development of software in the engineering and design areas.
- business process re-engineering: A process model with specific information items can be used for reengineering of an organization like other process models.

Also any form of a data model defined in EXPRESS can be read into GTPPM as an *information menu*. Using this function, GTPPM can be used to update or validate an existing product model by reading in an existing product model as an information menu. It can be also used to develop *conformance classes* (i.e., valid subset models) of an existing model.

GT PPM has been implemented as a Microsoft Visio® Add-on. The tool has been applied to fourteen companies of the North American Precast Concrete Software Consortium (PCSC) and is being applied to three IT-related research projects at Purdue, Carnegie Mellon, and Teeside University (UK). Experience to date indicates that GT PPM holds the potential to improve and expedite product model development.

The author believes that a newly proposed *process to product modeling* method and its supporting procedures provide the logic and a promising means to (semi-)automatically derive a product model from collected process information.

APPENDIX A
EARLY STANDARD PRODUCT MODELING EFFORTS

This appendix summarizes early standard product modeling efforts (Goldstein, Kemmerer, and Parks 1998) (Bloor and Owen 1995):

Table 8.1 Chronology of development in product data

STEP					STEP→				DPI	Initial parts				11.31 approved	Initial release
US					PDDI		PDES Initiation		PDES Inc→						
IGES	IGES 1.0			IGES 2.0			IGES 3.0		IGES 4.0		IGES 5.0	IGES 5.1		IGES 5.2	
Subsets								MIL-D-28000					MIL-D-28000A		
Germany					VDA-FS 1.0		DIN	VDA-IS 1.0 VDA-FS 2.0		VDA-IS 2.0					
France					SET 1.1	afnor				afnor					
	1980	**1981**	**1982**	**1983**	**1984**	**1985**	**1986**	**1987**	**1988**	**1989**	**1990**	**1991**	**1992**	**1993**	**1994**
Europe					ESPRIT→					ESPRIT II→				ESPRIT III→	
Testing		Autofact →				CTS→	NAVFAC			CTS-2→					
NEDO					NEDO1		NEDO2								
Graphics						GKS		CGM	GKS-3D	PHIGS		CGI	PHIGS-PLUS CGRM		
EDIF						EDIF 100		EDIF 200						EDIF 300	
Modeling		IDEF0-2	NIAM			IDEF1x							IDEF3-4		

(Source: (Bloor and Owen 1995))

- the ANSI/X3/SPARC methodology: The X3/SPARC Committee of the American National Standards Institute (ANSI) developed the three-layer (conceptual, internal, external layers) architecture of information modeling.
- ANSI Y14.26 (Digital Representation for Communication of Product Definition Data, 1970-1981): is an ANSI committee for standardization of a product model.
- CAM-I (1973-1984): the Computer-Aided Manufacturing – International Inc. (CAM-I) organization significantly contributed to the formal description of Boundary Representation (BRep).

- IGES (1979-1981): IGES (Initial Graphics Exchange Specification) provided the first practical solution for CAD data exchange with an exchange file format.
- the ICAM Program: The Integrated Computer Aided Manufacturing (ICAM) program, funded by the U.S. Air Force, developed the IDEF method for process and information modeling.
- AECMA Report of geometry data exchange study group: The European Association of Aerospace Industries (AECMA) developed a standard data format for exchanging surface geometry.
- the VDA in 1982: Flachenschnittstelle Des Verbandes Der Deutschen Automobilindustrie (VDA-FS and VDA-IS) is German efforts to develop a standard data model for exchanging drawing information, two- and three-dimensional geometry, analytic and free form surfaces/curves required for the automotive industry.
- the SET project in 1983: Pure geometric data models such as IGES has been criticized for not being able to describe the full lifecycle of a product. The French Standard d'Echange et de Transfert (SET) project has been continued by Association GOSET, which became contributors to ISO 10303 and STEP conformance testing services.
- the Product Definition Data Interface (PDDI, 1982-1987): The PDDI was a research projected funded by the ICAM program to develop a method to exchange and share geometric data among computer applications without human intervention based on an thorough evaluation of IGES (ANSI: Product Definition Data Interface 1983).

- NBS: National Bureau of Standards (NBS, currently NIST), sponsored by the U.S. Department of Defense Computer-Aided Acquisition and Lifecycle Support (CALS) program, led the development of IGES subsets. STEP's concept of application protocols (APs) and Conformance Classes grew from this and other early work.
- ISO TC 184/SC4 Meeting (1984): International Organization for Standardization (ISO) STEP (STandard for Exchanging Product (data) model) began in 1984.
- CTS (since 1985): The Conformance Test Suite (CTS) project is a project to develop conformance-testing methods and to establish testing services ((Bloor and Owen 1995) p.141).
- the Product Data Exchange Specification (PDES, 1984-1985): In 1984, the PDES has been proposed as the next generation of IGES and as a response to the PDDI and other European standardization efforts to support the full lifecycle of products and more complex products and software environment.
- MIL-D specifications (1987): the subsets developed by the US Department of Defense (DoD)
- ESPRIT: the EU information technologies program (http://www.cordis.lu/esprit/home.html)
- US Harmonization of Product Data Standards Organization (1989): NIST was the leader of the Harmonization of Product Data Standards (HPS) organization under the Industrial Automation Planning Panel (IAPP) of ANSI. The intent of HPS was to derive a harmonized Application Reference Model (ARM) from several U.S. standards (e.g., IPC, IGES/PDES, IEEE, EIA) and to integrate them with STEP.

APPENDIX B
THE FORMAL DEFINITION OF THE SEMANTIC UNION

A *semantic union* is different from a simple aggregation of data sets or a general union. It can be formally defined as:

$$A \cup^* B = (A + B) - [(A \cap^+ B) - (A \cap^* B)]$$

$\cup^*$: semantic union

$\cap^+$: a set (or aggregation) of semantically equivalent entities

$\cap^*$: semantic intersection

where *A* and *B* are respectively a set of data required by an application or a work process.

If we use the same example from Section 3.1,

$A \equiv$ {project_name, load, driver}

$B \equiv$ {strucutre_name, load, frame}

$A + B \equiv$ {project_name, structure_name, load, load, driver, frame}

$A \cap B \equiv$ {load}

$A \cup B \equiv$ {project_name, load, driver, structure_name, frame}

Let project_name in Set *A* be a *synonym* of structure_name in Set *B*

load (truck load) in Set *A* is a *homonym* of load (structural load) in Set *B*

In such a case, the results of the semantic set operations of these two sets will be:

$A + B \equiv$ {project_name, structure_name, load, load, driver, frame}

$A \cap^{+} B \equiv$ {project_name, structure_name}

$A \cap^{*} B \equiv \{ F_{si}$(project_name, structure_name)}

where F_{si}(x, y): returns an semantic intersection of elements x and y

Let f_{si}(project_name, structure_name) = project_name

$A \cap^{*} B \equiv$ { project_name}

$(A \cap^{+} B) - (A \cap^{*} B) \equiv$ {structure_name}

$\therefore A \cup^{*} B \equiv$ {project_name, load, load, driver, frame}

The definition of the semantic union can be simplified by introducing *complement intersection* $\cap^{c}$. The *complement intersection* $\cap^{c}$ can be defined as the subtraction of semantic intersection from a set of semantically equivalent entities similar to the *complement set*[24]:

$$A \cap^{c} B = (A \cap^{+} B) - (A \cap^{*} B) \text{ or } A \cap^{c} B \equiv \{x \mid x \in A \cap^{+} B, x \notin A \cap^{*} B\}$$

Using the complement intersection, the *semantic union* can be redefined as a subtraction of a *complement intersection of semantic intersection* of different native data models from an *aggregation* of the data sets, similar to the definition of a general union[25].

$$A \cup^{*} B = (A + B) - (A \cap^{c} B)$$

[24] Complement Set of *C*, $C^{c} \equiv \{x \mid x \in S, x \notin C, C \subseteq S\}$

[25] Union Set $A \cup B = (A + B) - (A \cap B)$

Since it is not possible that a software application can automatically recognize homonyms or synonyms without any additional information, it is obvious that two instances of ‘load’ in the above example should be replaced by distinguishable terms in a practical model. For example,

D ≡{project_name, truck_load, structural_load, driver, frame}

APPENDIX C
RESOURCES FOR PROCESS MODELING METHODS

A.1 OVERVIEW

This appendix summarizes resources for major process modeling techniques today.

A.2 A BRIEF HISTORY OF PROCESS MODELING

Even though some literatures claims that process management has existed since prehistoric times, it is a general view to regard Frederick Taylor (1919) as a father of the modern process management (Eastman and Shirley 1994; Osborne and Nakamura 2000). The historic evolution of process modeling methods -from early Gantt charts (1955) and PERT/CPM to modern structured analysis by Tome DeMacro (the 1980s) - are well reviewed by Osborne (Osborne and Nakamura 2000, Ch 2). In the early 1990s, data-centered, scenario-based, structural methods were synthesized into one modeling language, which became the current United Modeling Language (UML).

A.3 RESOURCES FOR MODELING METHODS AND EXCERPTS FROM THEM

This section lists electronic resources for major process modeling methods and provides a short excerpt on the modeling method from the webpage. Excerpts are in *italic*.

- IDEFØ:
  http://www.IDEF.com

IDEFØ is a method designed to model the decisions, actions, and activities of an organization or system. IDEFØ was derived from a well-established graphical language, the Structured Analysis and Design Technique (SADT). The United States Air Force commissioned the developers of SADT to develop a function modeling method for

analyzing and communicating the functional perspective of a system. Effective IDEFØ models help to organize the analysis of a system and to promote good communication between the analyst and the customer. IDEFØ is useful in establishing the scope of an analysis, especially for a functional analysis. As a communication tool, IDEFØ enhances domain expert involvement and consensus decision-making through simplified graphical devices. As an analysis tool, IDEFØ assists the modeler in identifying what functions are performed, what is needed to perform those functions, what the current system does right, and what the current system does wrong. Thus, IDEFØ models are often created as one of the first tasks of a system development effort.

In December 1993, the Computer Systems Laboratory of the National Institute of Standards and Technology (NIST) released IDEFØ as a standard for Function Modeling in FIPS Publication 183.

- Petri Net
    - Tutorial: http://worldserver.oleane.com/adv/elstech/petrinet.htm
    - Petri Net World: http://www.daimi.au.dk/PetriNets/
    - Tools: http://www.daimi.au.dk/PetriNets/tools/quick.html
    - CPN: http://www.daimi.au.dk/CPnets/
    - Dr. Carl Adam Petri:

      http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri_eng.html

(Excerpt from *http://worldserver.oleane.com/adv/elstech/petrinet.htm*)

Petri nets were introduced by C.A.Petri in the early 1960s as a mathematical tool for modeling distributed systems and, in particular, notions of concurrency, non-determinism, communication and synchronization. Their further development was

facilitated by the fact that Petri Nets easy model process synchronization, asynchronous events, concurrent operations, and conflicts or resource sharing. Petri Nets have been successfully used for concurrent and parallel systems modeling and analysis, communication protocols, performance evaluation and fault-tolerant systems.

- DFD

  (a.k.a Yourdon)
  - http://spot.colorado.edu/~kozar/DFD.html
  - http://www.doc.mmu.ac.uk/online/SAD/T04/dfds.htm
  - http://www.aisintl.com/case/drd.html

  (Excerpts from http://spot.colorado.edu/~kozar/DFD.html)

  Data flow diagrams are a network representation of a system. They are the cornerstone for structured systems analysis and design. The diagrams use four symbols to represent any system at any level of detail. The four entities that must be represented are:

  - *data flows - movement of data in the system*
  - *data stores - data repositories for data that is not moving*
  - *processes - transforms of incoming data flow(s) to outgoing data flow(s)*
  - *external entities - sources or destinations outside the specified system boundary*

  Data flow diagrams do not show decisions or timing of events. Their function is to illustrate data sources, destinations, flows, stores, and transformations. The capabilities of data flow diagramming align directly with general definitions of systems. Data flow diagrams are an implementation of a method for representing systems concepts including boundaries, input/outputs, processes/subprocesses, etc.

The data flow diagram is analogous to a road map. It is a network model of all possibilities with different detail shown on different hierarchical levels. The process of representing different detail levels is called "leveling" or "partitioning" by some data flow diagram advocates.

- SSADM (Structured Systems Analysis and Design Methodology) Diagrams
  http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html

  SSADM (in common with other structured methodologies) adopts a prescriptive approach to information systems development in that it specifies in advance the modules, stages and tasks which have to be carried out, the deliverables to be produced and furthermore the techniques used to produce the deliverables. SSADM adopts the Waterfall model of systems development, where each phase has to be completed and signed off before subsequent phases can begin.

- STRADIS: (Structured Analysis, Design and Implementation of Information Systems)
  http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html

  A methodology developed by Gane and Sarson (1979). The methodology is based on the philosophy of top down functional decomposition and relies on the use of Data Flow Diagrams.

- YSM: (Yourdon Systems Method,Yourdon, 1993)
  http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html

  YSM is similar to STRADIS in its use of functional decomposition, however a middle-out approach is dopted and slightly more emphasis is placed on the importance of data structures.

- MERISE: (Quang and Chartier-Kastler, 1991)
  http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html

The methodology is widely used in ISE in France, Spain and Switzerland. MERISE consists of three 'cycles', the decision cycle, the life cycle and the abstraction cycle. The abstraction cycle is the key, in this cycle both data and processes are viewed firstly at the conceptual level, then the logical or organizational level and finally at the physical or operational level.

- EUROMETHOD: (CCTA, 1994)
  http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html

Euromethod could be described as a framework for the integration of existing european methodologies rather than as a methodology in its own right.

- The UML (Unified Modeling Language)
  http://www.rational.com (or http://www-306.ibm.com/software/rational/)
  http://www.omg.org/UML
  http://uml.shl.com

(Excerpt from *http://www.omg.org/UML*)

The OMG's Unified Modeling Language™ (UML®) helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements. (You can use UML for business modeling and modeling of other non-software systems too.) Using any one of the large number of UML-based tools on the market, you can analyze your future application's requirements and design a solution that meets them, representing the results using UML's twelve standard diagram types.

A.4 RELATIONS BETWEEN PROCESS MODELING METHODS

- SADT (Structural Analysis and Design Techniques)

  IDEFØ was derived from SADT.

- SSADM (Structured Systems Analysis and Design Methodology)

  SSADM and SADT are not the same. (http://www.csci.csusb.edu/dick/methods.html)

- UML (Unified Modeling Languaage)

  The UML was built on three major streams of modeling methods (Rosenberg and Scott 1999, Ch 1).

Table 8.2. Three major streams of the UML

Data-Centered Method	Scenario-Based Methods	Structural Methods
SADT Shlaer/Mellor Martin/Odell Rumbaough's OMT ERDs DFDs State-Transition diagrams	Jascobson's OOSE, Use-case driven approach ParcPlace – OBA Alger/Goldstein – Scenario based Method	OO Programming Booch Method Wirfs-Brock's CRC Cards

The UML are composed of twelve standard diagrams (Booch, Rumbaugh, and Jacobson 1999).

Table 8.3. Twelve standard UML Diagrams

Diagram Type	Diagram	Definition
Structural Diagrams	Class Diagram	shows a set of classes, interfaces, and collaborations and their relationships.
	Object Diagram	shows a set of objects and their relationships.
	Component Diagram	shows the organizations and dependencies among a set of components.
	Deployment Diagram	shows the configuration of run-time processing nodes and the components that live on them.

Table 8.3 Twelve standard UML Diagrams (continued)

Diagram Type	Diagram	Definition
Behavior Diagrams	Use Case Diagram	shows a set of use cases and actors and their relationship.
	State Diagram (Statechart Diagram)	shows a state machine, consisting of states, transitions, events, and activities.
	Activity Diagram	shows the flow from activity to activity within a system. An activity shows a set of activities, the sequential or branching flow from activity to activity, and objects that act and are acted upon.
	Sequence Diagram	is an interaction diagram that emphasizes the time ordering of messages
	Collaboration Diagram	is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. A collaboration diagram shows a set of objects, links among those objects, and messages sent and received by those objects.
Model Management Diagrams	Package	A general-purpose mechanism for organizing elements into groups
	Subsystem	A grouping of elements of which some constitute of a specification of the behavior offered by the other contained elements
	Model	A simplification of reality, created in order to better understand the system being created; a semantically closed abstraction of a system

The Unified Modeling Language (UML) literally includes most of major modeling languages today and is still evolving. (http://www.omg.org)

- Petri Net

  Colored Petri Net (CPN) is variation of the traditional Petri Net.

- Flowchart

  The Flowchart method is an ANSI standard (ANSI-IEEE standard 5807-1985 (ANSI 1991; Osborne and Nakamura 2000, Ch 6)).

A.5 PROCESS MODELING TOOLS IN THE MARKET

Table 8.4 shows some of commercial process modeling tools available today and their developers.

Table 8.4. Process modeling tools

Name	a.k.a	Company/Developer
4Keeps		4Keeps, Inc (Former A.D. Experts)
ActiveModeler		Kaisha-tec
AI0 Win60		KBSI
AllFusion (Process Modeler)	BPWin	Computer Associates (Former Plantinum)
ARIS Toolset		IDS Scheer
Enterprise Modeller		Business Integration Technologies
Hyperformix Workbench	SES/workbench	Hyperformix
iGrafx 2000		Micrografx
MooD		Morphix
ProcessWise Workbench		Fujitsu Teamware
SmartDraw		SmartDraw.com
Arena		Rockwell Software (Systems Modeling Corporation)
BPD	Lifecycle Manager	Qualiware
Corporate Modeler		Casewise
CRISP-DM	CRISP 1.0	SPSS
iThink		Cognitus
Metify ABM		Armstrong Laing Group
Oracle Designer		Oracle
ProcessModel	http://www.processmodel.com/	ProcessModel
ProSim6.0		KBSI
SmartER		KBSI
Visio		Microsoft
WorkFlow Modeler		Meta Software

A.6 ORGANIZATIONS RELATED TO PROCESS MODELING

Several non-profitable organizations exist to develop standards and integrate process modeling efforts. The organizations and their self-introductions are as follows:

- Work flow Management Coalition (WfMC)

  http://www.wfmc.org/

The WfMC has over 300 member organizations worldwide, representing all facets of workflow, from vendors to users, and from academics to consultants.

Subgroup: e-Workflow, http://www.e-workflow.org/

- Workflow and Reengineering International Association (WARIA)

  http://www.waria.com/

  The charter of the Workflow And Reengineering International Association (WARIA) is to identify and clarify issues that are common to users of workflow, electronic commerce and those who are in the process of reengineering their organizations. The association facilitates opportunities for members to discuss and share their experiences freely. Established in 1992, WARIA's mission is to make sense of what's happening at the intersection of Business Process Management, Workflow, Knowledge Management and Electronic Commerce and reach clarity through sharing experiences, product evaluations, networking between users and vendors, education and training.

- The Business Process Modeling Language (BPMI)

  http://www.bpmi.org/

  BPMI.org (the Business Process Management Initiative) is a non-profit corporation that empowers companies of all sizes, across all industries, to develop and operate business processes that span multiple applications and business partners, behind the firewall and over the Internet. The Initiative's mission is to promote and develop the use of Business Process Management (BPM) through the establishment of standards for process design, deployment, execution, maintenance, and optimization. BPMI.org

develops open specifications, assists IT vendors for marketing their implementations, and supports businesses for using Business Process Management technologies.

- The Association for Information and Image Management (AIIM)

  http://www.aiim.org

  A lot has changed since AIIM (The Association for Information and Image Management) was founded in 1943 as the National Microfilm Association. But one thing has remained remarkably consistent. Despite countless revolutions in technologies, our core focus has remained the same -- helping users connect with suppliers who can help them apply document and content technologies to improve their internal processes. AIIM International is the industry's leading global organization. We believe that at the center of an effective business infrastructure in the digital age is the ability to capture, create, customize, deliver, and manage enterprise content to support business processes. The requisite technologies to establish this infrastructure are an extension of AIIM's core document and content technologies. These Enterprise Content Management (ECM) technologies are key enablers of e-Business and include: Content/Document Management, Business Process Management, Enterprise Portals, Knowledge Management, Image Management, Data Warehousing, and Data Mining. Our focus over the next 3-5 years will be helping our members - both users and suppliers – make this e-Business transition.

- BizTalk

  http://www.biztalk.org/

The goal of BizTalk.org is to provide resources for learning about and using Extensible Markup Language (XML) for Enterprise Application Integration (EAI) and business-to-business (B2B) document exchange, both within the enterprise and over the Internet. On BizTalk.org you can learn how to use XML messages to integrate software applications and build new solutions. The design emphasis is to use XML to integrate your existing data models, solutions, and application infrastructure, and adapt them for electronic commerce. You can also learn about the BizTalk Framework, a set of guidelines for implementing an XML schema and a set of XML tags used in messages sent between applications.

- ebXML

  http://www.ebxml.org

  To provide an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure and consistent manner by all parties.

- Object Management Group (OMG)[26]

  http://www.omg.org/

  The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. Our membership roster, about 800 strong, includes virtually every large company in the computer industry, and hundreds of smaller ones. Most of the companies that shape enterprise and Internet computing today are represented on our Board of Directors. Our flagship specification is the multi-platform Model Driven Architecture (MDA), recently underway but already well

[26] The OMG is the official group which maintains the UML.

known in the industry. It is based on the modeling specifications the MOF, the UML, XMI, and CWM. OMG's own middleware platform is CORBA, which includes the Interface Definition Language OMG IDL, and protocol IIOP. The Object Management Architecture (OMA) defines standard services that will carry over into MDA work shortly. OMG Task Forces standardize Domain Facilities in industries such as healthcare, manufacturing, telecommunications, and others.

A.7 OTHER RESOURCES FOR PROCESS MODELING

- SODAN: http://www.sodan.co.uk/main.html?s=modeling

  SODAN sells an overview of workflow and process modeling tool products and suppliers (£375/each).

- Bart-Jan Hommes: http://is.twi.tudelft.nl/~hommes/toolsub.html

- A Glossary of Software Development Methods:

  http://www.csci.csusb.edu/dick/methods.html

  Dick Botting provides short definitions of over 100 software development methods and terms.

- Evaluation of Systems Analysis Methodologies in a Workflow Context

  http://computing.unn.ac.uk/staff/cgnr1/badensoft.htm

  Fahad Al-Humaidan and B. Nick Rossiter compare OPM, SSADM, UML, Unified Process, SSM, and WfMS in fourteen categories.

APPENDIX D
REQUIREMENTS COLLECTION METHODS

Requirements collection activities rely on a variety of formalisms including Flowcharts, UML Activity Diagrams, the Use Case diagrams, Data Flow Diagrams (DFDs) (Osborne and Nakamura 2000), and IDEF0 (NIST 1993) schemas. Both Flowcharts (ANSI 1991) and Activity Diagrams (Booch, Rumbaugh, and Jacobson 1999) are limited only to capturing sequences of activities and are not able to describe the information used in a process. Use Case diagrams (Jacobson, Jonsson, and Overgaard 1992) which are a part of the UML methodology, define a set of sequences in which each sequence represents the interaction of the things outside the system (its actors) with the system itself (and its key abstractions) (Booch, Rumbaugh, and Jacobson 1999). Data flow Diagrams (DFDs) (Osborne and Nakamura 2000) consists of several levels of diagrams. The top-level DFD is called a *context diagram*. Details of information that is transferred between processes and data storages is separately described and called a *data dictionary*. However, DFDs do not show workflows, i.e., decisions or sequences of activities. DFDs capture information required for 'system' design, but do not describe information flows in a sequence of activities.

IDEF0 (Integration Definition of Function Modeling is a Federal Information Processing Standard (FIPS) supported by ISO and is designed to define the "functions of a system or subject area with graphics, text and glossary (NIST 1993)." As in DFD modeling, IDEF0 models have a hierarchical structure and take a top-down approach. A unique feature of IDEF0 is its ICOM codes (Input, Control, Output, and Mechanism arrows). Although arrow types are categorized in detail, IDEF0 tracks information in chunks, but

not in terms of individual information items. Detailed information can be defined separately in IDEF1x (or IDEF1), but there is no direct link between the two modeling techniques.

The above modeling methods are incorporated into a set of commercial tools (e.g., BPR®, Arena®, Rose®, and SmartDraw®). They have been further researched and enhanced in several systems: (e.g., PetriNet (Benwell, Firns, and Sallis 1991; Petri 1962), OSMOS (Wilson et al. 2001), GPP (Wix and katranuschkov 2002), ISTforCE (Wix and Liebich 2000), ATLAS (Tolman and Poyet 1995), and ICCI (Katranuschkov et al. 2002)) have been developed, to enhance or integrate existing modeling methods. Some commercial CASE (Computer-Aided Software Engineering) tools for database design (such as Visio®, AllFusion® (a.k.a. ERWin®, BPWin®, ModelMart®)), and Corporate Modeler®) are capable of coupling DBMSs mostly with ARMs (e.g., IDEF1x, EXPRESS-G, and ER diagrams) and sometimes with process models (AAMs). However, they do not provide any formal method to elicit information from heterogeneous business environments and to integrate the collected information into an industry-level product model.

Appendix C provides additional information and resources on process modeling methods.

APPENDIX E
NOTATION OF A CONTEXT-FREE GRAMMAR (CFG)

The *context-free grammar* (CFG) is a formal system to define how any legal statement of a language can be derived by a set of *axioms*. The axioms are the *rewrite rules* of a language. A syntax of the CFG is a duplex <B, R>, where B is the union of *terminals* and *non-terminals* and R is the set of axioms or rules. For example, 'W → χ' denotes a syntactic rule 'W can *be replaced by* χ.' The arrow (→) is called the *rewrite arrow* and reads 'is-a.' Note that W → χ is different from χ → W. 'W' must always be a non-terminal symbol and χ is a string of either a terminal or a non-terminal symbol. A terminal symbol is a lexical item that cannot be split into smaller constituents of a language. Examples are {a, black, cat, ran} in Figure 8.2. A non-terminal symbol is a non-lexical symbol that represents a class of terminal symbols. Examples include {S (subject), NP (noun phrase), VP (verb phrase), N (noun), V (verb), Det (determiner), Adj (adjective)} in Figure 8.2. '-' denotes concatenation of symbols.

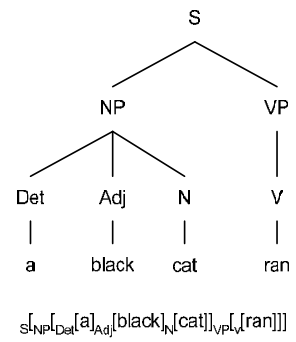


Figure 8.2. A linguistic example of a constituent structure tree

The context-free grammar can be depicted as a breakdown structure. The structure is called a *constituent structure tree*. The vertical breakdown denotes the *is-a* categorization like the arrow (→) and the horizontal enumeration represents grammatical relations of terminals and non-terminals such as *subject-of*, *object-of*, and *modifier*. Figure 8.2 is an example of a constituent structure tree of a sentence "A black cat ran."

The given rewrite rules for 'A black cat ran' are as follows:

S → NP – VP

NP→ Det – Adj – N

VP → V

Det → a

Adj → black

N → cat

V → ran

A → B | C denotes A → B or A → C.

In a context free grammar, the left side of a re-write rule is limited to a single non-terminal. The right side can be replaced by a null value in order to accommodate *abbreviation* or *replacement* phenomena.

W → NULL

For example, in English imperative, the subject "you" can be omitted:

S → NP – VP

NP → NULL

VP → go away

APPENDIX F

A PSEUDO CODE[27] FOR DETECTING SEMANTICALLY EQUIVALENT INFORMATION CONSTRUCTS

```
FUNCTION Is_Semantically_Equivalent_ICs
    DIM x as information_construct
    DIM y as information_construct
    DIM UnabbreviatedIC as information_construct
    DIM AbbreviatedIC as information_construct

    IF len(x) = len(y) THEN
        IF x = y THEN
            Is_Semantically_Equivalent_ICs = TRUE
            Merge_the_attributes_of_x_and_y_into_x
        ELSE
            Is_Semantically_Equivalent_ICs = FALSE
        END IF
    ELSE
        IF len(x) > len(y) THEN
            UnabbreviatedIC = x
            AbbreviatedIC = y
        ELSE
            UnabbreviatedIC = y;
            AbbreviatedIC = x;
        END IF
        IF left(UnabbreviatedIC, len(AbbreviatedIC)+1) = "*" + _ AbbreviatedIC
            Is_Semantically_Equivalent_ICs = TRUE
            Merge_the_attributes_of_x_and_y_into_UnabbreviatedIC
            Delete_AbbreviatedIC_and_its_attributes
        ELSE
            Is_Semantically_Equivalent_ICs = FALSE
        END IF
    END IF
```

[27] The pseudo code follows the Visual Basic grammar.

```
END FUNCTION


SUB Merge_the_attributes_of_x_and_y_into_x
END SUB


SUB Merge_the_attributes_of_x_and_y_into_UnabbreviatedIC
END SUB


SUB Delete_AbbreviatedIC_and_its_attributes
END SUB
```

APPENDIX G

AUTOMATICALLY GENERATED PRELIMINARY PRODUCT MODELS IN EXPRESS

```
SCHEMA unihighcti_042704;


ENTITY documentation
    SUPERTYPE OF (ONEOF(
    drawing,
    bom,
    bidding_documents)
    );
    qc_check: qc_check;
    report_date: string;
    to_be_sent_to: string;
    revision_no: string;
    revised_date: string;
    report_time: string;
    revised_by: string;
    received_date: string;
    requirements: string;
    id: string;
END_ENTITY;


ENTITY piece
    SUPERTYPE OF (ONEOF(
    floor_piece,
    spandrel,
    pc_column)
    );
    pack: pack;
    windows: windows;
    bowing: bowing;
    design_requirements: design_requirements;
    wythe: wythe;
```

```
    sample: sample;
    estimation: estimation;
    material: material;
    mold: mold;
    geometry: geometry;
    surface_treatment: surface_treatment;
    shipping: shipping;
    piece_mark: string;
    product_unit_measurement: string;
    product_size: string;
    product_amount: string;
    product_name: string;
    product_code: string;
    label: string;
    mobilization: string;
    blockout: blockout;
    hardware_list: hardware_list;
    connection: connection;
    location_details: location_details;
    production_and_handling: production_and_handling;
    drawing: drawing;
END_ENTITY;

ENTITY floor_piece
    SUPERTYPE OF (ONEOF(
    dt)
    )
    SUBTYPE OF (
    piece
    );
END_ENTITY;

ENTITY pc_column
    SUPERTYPE OF (ONEOF(
    exterior_pc_column)
    )
```

```
    SUBTYPE OF (
    piece
    );
END_ENTITY;

ENTITY drawing
    SUPERTYPE OF (ONEOF(
    piece_drawing,
    erection_drawing,
    section_drawing,
    plan_drawing,
    detail_drawing,
    elevation_drawing,
    foundation_drawing)
    )
    SUBTYPE OF (
    documentation
    );
    sealed: string;
    created_date: string;
    created_by: string;
    destroyed_date: string;
    engineering: engineering;
    callout: string;
    due_date: string;
END_ENTITY;

ENTITY assembly
    SUPERTYPE OF (ONEOF(
    floor_assembly)
    );
    dimensions: dimensions;
    piece_list: piece_list;
    grid: grid;
END_ENTITY;
```

```
ENTITY connection
    SUPERTYPE OF (ONEOF(
    cip_haunch,
    tieback,
    corbel,
    pocket,
    dap,
    chord)
    );
    requirement: string;
    material: material;
    light_pole: light_pole;
    erection_sleeve: erection_sleeve;
    reinforcement: reinforcement;
    spacing: string;
    xx_type: string;
    piece_list: piece_list;
END_ENTITY;

ENTITY geometry
    SUPERTYPE OF (ONEOF(
    geometry_2d,
    geometry_3d)
    );
    id: string;
    constraints: constraints;
    geometry_3d: geometry_3d;
    dimensions: dimensions;
END_ENTITY;

ENTITY hardware
    SUPERTYPE OF (ONEOF(
    grout,
    tieback,
    handling_bolt,
    bolt,
```

```
    handling_insert,
    handling_leg,
    erection_anchor,
    custom_item,
    cast_in_box,
    curtainwall_insert,
    shipping_frame,
    temporary_bracing,
    anchor,
    plate,
    lifting)
    )
    SUBTYPE OF (
    material
    );
    surface_treatment: surface_treatment;
END_ENTITY;

ENTITY reinforcement
    SUPERTYPE OF (ONEOF(
    prestressing,
    rebar,
    mesh)
    )
    SUBTYPE OF (
    material
    );
    youngs_modulus: string;
    waste: string;
    bpc_end_geometry: string;
    crosssectional_area: string;
    qc_check: qc_check;
    location_details: location_details;
END_ENTITY;

ENTITY qc_check
```

```
    SUPERTYPE OF (ONEOF(
    interference_check)
    );
    inspector_id: string;
    id: string;
    requirements: string;
    inspector: string;
    inspection_dates: string;
    building_code: building_code;
    results: string;
END_ENTITY;

ENTITY rebar
    SUPERTYPE OF (ONEOF(
    bent_bar)
    )
    SUBTYPE OF (
    reinforcement
    );
    temperature: string;
    diameter: string;
END_ENTITY;

ENTITY material
    SUPERTYPE OF (ONEOF(
    hardware,
    caulk,
    rigging,
    handling,
    offsite_staging,
    erection_handling_frame,
    gutter_system,
    strongback,
    brick,
    stone,
    electric_items,
```

```
    reinforcement,
    retarder,
    admixture,
    pigment,
    pc_aggregate,
    form_panelization,
    form_handling,
    form_liner,
    adhesive,
    back_forming,
    concrete)
    );
    purchase_order: purchase_order;
    unit_price: string;
    pattern: string;
    id: string;
    xx_type: string;
    quantity: string;
END_ENTITY;

ENTITY production_and_handling
    SUPERTYPE OF (ONEOF(
    welding,
    repair,
    batch,
    pour)
    );
    pour: pour;
    equipment: equipment;
    operation_details: string;
    operation_cost: string;
    cost: string;
    production_cost: string;
    production_per_hour: string;
    weather: string;
    yard_cost: string;
```

```
    electric_power_req: string;
    manager: string;
    concrete: concrete;
    labor: labor;
    schedule: schedule;
END_ENTITY;

ENTITY schedule
    SUPERTYPE OF (ONEOF(
    erection_schedule)
    );
    piece_list: piece_list;
    approved_date: string;
    schedule_date: string;
    actual_start_date: string;
    actual_pc_end_date: string;
    required_duration: string;
    planned_duration: string;
    planned_start_date: string;
    planned_pc_end_date: string;
END_ENTITY;

ENTITY erection_schedule
    SUPERTYPE OF (ONEOF(
    installation_schedule,
    foundation_schedule)
    )
    SUBTYPE OF (
    schedule
    );
END_ENTITY;

ENTITY equipment
    SUPERTYPE OF (ONEOF(
    pretension_gun,
    crane,
```

```
    roof_deck,
    safety_cable)
    );
    unit_cost: string;
    END_ENTITY;

    ENTITY geometry_2d
    SUPERTYPE OF (ONEOF(
    crosssection)
    )
    SUBTYPE OF (
    geometry
    );
    base_point: string;
END_ENTITY;

ENTITY grid;
    y_axis_spacing: string;
    x_axis_spacing: string;
END_ENTITY;

ENTITY dt
    SUBTYPE OF (
    floor_piece
    );
    qc_check: qc_check;
    mesh: mesh;
    pc_end: pc_end;
    flange: flange;
    recess: recess;
    chord: chord;
    dap: dap;
    structural_analysis: structural_analysis;
    stem: stem;
    stem_spacing: string;
    joint: joint;
```

```
END_ENTITY;

ENTITY joint;
    dimensions: dimensions;
END_ENTITY;

ENTITY dimensions;
    depth: string;
    total_length: string;
    total_poured_length: string;
    id: string;
    floor_to_floor_height: string;
    thickness: string;
    xx_length: string;
    height: string;
    cast_length: string;
    width: string;
END_ENTITY;

ENTITY exterior_pc_column
    SUBTYPE OF (
    pc_column
    );
    clearance: string;
    rebar: rebar;
    rebar_cage: rebar_cage;
    pocket: pocket;
    corbel: corbel;
    geometry_3d: geometry_3d;
    foundation_drawing: foundation_drawing;
    elevation_drawing: elevation_drawing;
    detail_drawing: detail_drawing;
    geometry_2d: geometry_2d;
    plan_drawing: plan_drawing;
END_ENTITY;
```

```
ENTITY plan_drawing
    SUBTYPE OF (
    drawing
    );
END_ENTITY;

ENTITY project;
    site: site;
    documentation: documentation;
    shipping: shipping;
    sales_representative: string;
    phase: string;
    job_number: string;
    size: string;
    owner_details: string;
    name: string;
    project_manager: string;
    contact_information: string;
    contractor_type: string;
    contractor_list: string;
    xx_type: string;
    contract_details: string;
    subctract_unit_cost: string;
    job_manager: string;
    approved_date: string;
    accountant: string;
    engineering_coorinator: string;
    design_requirements: design_requirements;
    estimation: estimation;
END_ENTITY;

ENTITY estimation;
    total_bid_prodice: string;
    unit_cost: string;
    item_code: string;
    item_description: string;
```

```
    item_quantity: string;
    id: string;
    unit_measurement: string;
    estimator: string;
    taxes: string;
    total_loads: string;
    schedule: string;
    gross_margin: string;
    total_markup: string;
END_ENTITY;

ENTITY location_details;
    orientation: string;
    base_point: string;
END_ENTITY;

ENTITY stem;
    geometry: geometry;
    mesh: mesh;
    spacing: string;
END_ENTITY;

ENTITY section_drawing
    SUBTYPE OF (
    drawing
    );
END_ENTITY;

ENTITY floor_assembly
    SUBTYPE OF (
    assembly
    );
    wash: wash;
    pc_topping: pc_topping;
END_ENTITY;
```

```
ENTITY pc_topping;
    xx_type: string;
END_ENTITY;

ENTITY piece_list;
    piece: piece;
    xx_list: string;
    quantity: string;
    id: string;
END_ENTITY;

ENTITY structural_analysis;
    start_date: string;
    results: string;
END_ENTITY;

ENTITY dap
    SUBTYPE OF (
    connection
    );
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY geometry_3d
    SUBTYPE OF (
    geometry
    );
    reinforcement: reinforcement;
    weight: string;
    volume: string;
    bottom: string;
    pc_top: string;
    xx_type: string;
    design: string;
END_ENTITY;
```

```
ENTITY hardware_list;
    hardware: hardware;
    id: string;
END_ENTITY;

ENTITY lifting
    SUBTYPE OF (
    hardware
    );
    standard_details: string;
    location_details: location_details;
END_ENTITY;

ENTITY chord
    SUBTYPE OF (
    connection
    );
    location_details: location_details;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY detail_drawing
    SUBTYPE OF (
    drawing
    );
END_ENTITY;

ENTITY elevation_drawing
    SUBTYPE OF (
    drawing
    );
END_ENTITY;

ENTITY foundation_drawing
    SUBTYPE OF (
    drawing
```

```
    );
END_ENTITY;

ENTITY corbel
    SUBTYPE OF (
    connection
    );
    blockout: blockout;
    geometry_3d: geometry_3d;
    rebar: rebar;
    location_details: location_details;
END_ENTITY;

ENTITY spandrel
    SUBTYPE OF (
    piece
    );
    geometry_3d: geometry_3d;
    xx_type: string;
END_ENTITY;

ENTITY recess;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY wash;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY surface_treatment;
    id: string;
    cleaning: string;
    details: string;
END_ENTITY;

ENTITY flange;
```

```
    geometry: geometry;
    connection: connection;
END_ENTITY;

ENTITY pc_end;
    connection: connection;
END_ENTITY;

ENTITY mesh
    SUBTYPE OF (
    reinforcement
    );
    dimensions: dimensions;
END_ENTITY;

ENTITY blockout;
    location_details: location_details;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY interference_check
    SUBTYPE OF (
    qc_check
    );
END_ENTITY;

ENTITY pocket
    SUBTYPE OF (
    connection
    );
    id: string;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY erection_sleeve;
    geometry_3d: geometry_3d;
```

```
END_ENTITY;

ENTITY light_pole;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY tieback
    SUBTYPE OF (
    hardware,
    connection
    );
END_ENTITY;

ENTITY bent_bar
    SUBTYPE OF (
    rebar
    );
    spacing: string;
END_ENTITY;

ENTITY rebar_cage;
    rebar: rebar;
END_ENTITY;

ENTITY plate
    SUBTYPE OF (
    hardware
    );
END_ENTITY;

ENTITY nonprecast_element;
    geometry_3d: geometry_3d;
END_ENTITY;

ENTITY anchor
    SUBTYPE OF (
```

```
    hardware
    );
    anchor_details: string;
END_ENTITY;

ENTITY design_requirements;
    id: string;
    fire_rating_requirements: string;
    access_requirements: string;
END_ENTITY;

ENTITY shipping;
    project: project;
    shipping_frame: shipping_frame;
    schedule: schedule;
    qc_check: qc_check;
    truck_number: string;
    packer: string;
    truck_driver: string;
    trailer_number: string;
    cost: string;
    traffic_control: string;
    traffic_control_permit: string;
    traffic_control_personnel: string;
    crew: string;
    orientation: string;
    permits: string;
    special_req: string;
    instruction: string;
    receiver: string;
    notes: string;
    truck_load: truck_load;
END_ENTITY;

ENTITY truck_load;
    constraints: constraints;
```

```
    designer: string;
    purchase_order_num: string;
    qty: string;
    id: string;
END_ENTITY;

ENTITY engineering;
    qc_check: qc_check;
    eng_date: string;
    sealed: string;
END_ENTITY;

ENTITY building_code;
    provision_reference: string;
END_ENTITY;

ENTITY erection_drawing
    SUBTYPE OF (
    drawing
    );
END_ENTITY;

ENTITY labor;
    xx_type: string;
    rate: string;
    hours: string;
END_ENTITY;

ENTITY site;
    address: string;
    map: string;
END_ENTITY;

ENTITY concrete
    SUBTYPE OF (
    material
```

```
    );
    temperature: string;
    strength: string;
    mix: mix;
    mix_specification: string;
END_ENTITY;

ENTITY erection;
    safety_cable: safety_cable;
    roof_deck: roof_deck;
    hardware_list: hardware_list;
    tolerance: string;
    control_lines: string;
    hoistbay_location: string;
    crane: crane;
    qc_check: qc_check;
    cost: string;
    schedule: schedule;
END_ENTITY;

ENTITY constraints;
    min_length: string;
    max_length: string;
    average_length: string;
    average_weight: string;
    id: string;
END_ENTITY;

ENTITY pour
    SUBTYPE OF (
    production_and_handling
    );
    constraints: constraints;
    bed: bed;
    quantity: string;
    status: string;
```

```
    area: string;
END_ENTITY;

ENTITY mold;
    purchase_order: purchase_order;
    back_forming: back_forming;
    description: string;
    xx_type: string;
    id: string;
    name: string;
    adhesive: adhesive;
    form_liner: form_liner;
    form_handling: form_handling;
    form_panelization: form_panelization;
    schedule: schedule;
    dimensions: dimensions;
END_ENTITY;

ENTITY bidding;
    estimation: estimation;
    bidders: string;
    review: string;
    xx_type: string;
END_ENTITY;

ENTITY bidding_documents
    SUBTYPE OF (
    documentation
    );
END_ENTITY;

ENTITY bom
    SUBTYPE OF (
    documentation
    );
    created_by: string;
```

```
END_ENTITY;

ENTITY batch
    SUBTYPE OF (
    production_and_handling
    );
END_ENTITY;

ENTITY prestressing
    SUBTYPE OF (
    reinforcement
    );
    tolerance: string;
    actual_elongation: string;
    guage_pressure: string;
    net_pull: string;
    splice_chuck: string;
    temperature_over_pull: string;
    temp_adjustment: string;
    dead_pc_end_seating: string;
    live_pc_end_seating: string;
    theo_elongation: string;
    design_data: string;
    xx_length: string;
    temp_diff_btw_conc_strand: string;
    yield_stress: string;
    pretension_gun: pretension_gun;
    tension: string;
END_ENTITY;

ENTITY piece_drawing
    SUBTYPE OF (
    drawing
    );
    issued_date: string;
END_ENTITY;
```

```
ENTITY foundation_schedule
    SUBTYPE OF (
    erection_schedule
    );
END_ENTITY;

ENTITY sample;
    id: string;
    req: string;
    range: string;
    size: string;
    schedule: schedule;
END_ENTITY;

ENTITY structure;
    erection: erection;
    structural_analysis: structural_analysis;
END_ENTITY;

ENTITY installation_schedule
    SUBTYPE OF (
    erection_schedule
    );
END_ENTITY;

ENTITY mix;
    qc_check: qc_check;
    material: material;
    stregnth: string;
    sheet: string;
    sample: sample;
END_ENTITY;

ENTITY pc_aggregate
    SUBTYPE OF (
```

```
    material
    );
END_ENTITY;

ENTITY purchase_order;
    status: string;
    id: string;
END_ENTITY;

ENTITY pigment
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY admixture
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY retarder
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY form_panelization
    SUBTYPE OF (
    material
    );
    requirement: string;
END_ENTITY;

ENTITY form_handling
    SUBTYPE OF (
```

```
    material
    );
    requirement: string;
END_ENTITY;

ENTITY form_liner
    SUBTYPE OF (
    material
    );
    requirement: string;
END_ENTITY;

ENTITY adhesive
    SUBTYPE OF (
    material
    );
    requirement: string;
END_ENTITY;

ENTITY back_forming
    SUBTYPE OF (
    material
    );
    requirement: string;
END_ENTITY;

ENTITY curtainwall_insert
    SUBTYPE OF (
    hardware
    );
END_ENTITY;

ENTITY cast_in_box
    SUBTYPE OF (
    hardware
    );
```

```
END_ENTITY;

ENTITY electric_items
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY stone
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY brick
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY custom_item
    SUBTYPE OF (
    hardware
    );
END_ENTITY;

ENTITY erection_anchor
    SUBTYPE OF (
    hardware
    );
    requirement: string;
END_ENTITY;

ENTITY wythe;
    strongback: strongback;
END_ENTITY;
```

```
ENTITY strongback
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY gutter_system
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY shipping_frame
    SUBTYPE OF (
    hardware
    );
END_ENTITY;

ENTITY handling_leg
    SUBTYPE OF (
    hardware
    );
    requirement: string;
END_ENTITY;

ENTITY handling_insert
    SUBTYPE OF (
    hardware
    );
    requirement: string;
END_ENTITY;

ENTITY bolt
    SUBTYPE OF (
    hardware
```

```
    );
    requirement: string;
END_ENTITY;

ENTITY handling_bolt
    SUBTYPE OF (
    hardware
    );
    requirement: string;
END_ENTITY;

ENTITY erection_handling_frame
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY offsite_staging
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY crane
    SUBTYPE OF (
    equipment
    );
    load: string;
    location_diagram: string;
    name: string;
    location: string;
    tower_crane_fillin: string;
    hoist_bay_fillin: string;
    mat_req: string;
    communication_system: string;
    hoist_req: string;
```

```
END_ENTITY;

ENTITY handling
    SUBTYPE OF (
    material
    );
    requirement: string;
END_ENTITY;

ENTITY rigging
    SUBTYPE OF (
    material
    );
    requirement: string;
END_ENTITY;

ENTITY bowing;
    adjustment: string;
END_ENTITY;

ENTITY temporary_bracing
    SUBTYPE OF (
    hardware
    );
END_ENTITY;

ENTITY repair
    SUBTYPE OF (
    production_and_handling
    );
END_ENTITY;

ENTITY welding
    SUBTYPE OF (
    production_and_handling
    );
```

```
END_ENTITY;

ENTITY caulk
    SUBTYPE OF (
    material
    );
END_ENTITY;

ENTITY cip_haunch
    SUBTYPE OF (
    connection
    );
END_ENTITY;

ENTITY grout
    SUBTYPE OF (
    hardware
    );
    requirement: string;
END_ENTITY;

ENTITY roof_deck
    SUBTYPE OF (
    equipment
    );
    req: string;
END_ENTITY;

ENTITY safety_cable
    SUBTYPE OF (
    equipment
    );
    req: string;
    quantity: string;
END_ENTITY;
```

```
ENTITY windows;
    id: string;
END_ENTITY;

ENTITY curtainwall;
    id: string;
END_ENTITY;

ENTITY bed;
    id: string;
    movement: string;
END_ENTITY;

ENTITY pretension_gun
    SUBTYPE OF (
    equipment
    );
    id: string;
END_ENTITY;

ENTITY pack;
    num: string;
END_ENTITY;

ENTITY crosssection
    SUBTYPE OF (
    geometry_2d
    );
    polyline: string;
END_ENTITY;


END_SCHEMA; (* end of unihighcti_042704*)
```

APPENDIX H
WORKFLOW MANAGEMENT

Workflow management is “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (WFMC 1999).” It differs from pure process modeling in that it includes ‘execution’ and ‘management’ of business processes as well as their ‘specification’(Jablonski and Bussler 1996; Lawrence 1997; WFMC 1999). Workflow management systems control data flows (more often, documents flows) and specifies who is supposed to execute what action when. Examples include MQ Series Workflow® (IBM), BizFlow® (HandySoft), Workflow® (W4), i-Flow® (Fujitsu Software), and Staffware Process Suite® (Staffware). They are typically performed in heterogeneous and distributed work environments. Thus, some of directly relevant research areas naturally include distributed and mobile computing and data mining (such as OLAP (On-Line Analytical Processing) and data warehousing) that can enable users to inquire and view data from different points of view. Even though workflow management systems are similar to our work in that they combine processes and information flows, we regard workflow management as a separate vast area that deals with management and application of business processes and information and will not coincide with the focus of this project. We will, however, consider the formal workflow models that are mainly derived from transaction management in databases (Chakravarthy et al. 1990; Rusinkiewicz and Sheth 1995; Weikum 1991)

REFERENCES

Adachi, Y. (2002). Overview of IFC model server framework. *ECPPM 2002*, 367-372.

AISC. (2002). *CIMSteel Integration Standards Release 2 (CIS/2)*. Available: http://www.cis2.org/.

Alexander, C., S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. (1997). *A Pattern Language: Towns, Buildings, Construction*.New York: Oxford University Press.

Amor, R. (2001). Misconceptions about integrated project databases. *ITcon, 6*, 57-68.

ANSI. (1991). *American National Standards for Information Processing - Documentation Symbols and Conventions for Data, Program and System Flowcharts, Program Network Charts and System Resources Charts*.New York: American National Standards Institute.

ANSI: Product Definition Data Interface. (1983). *Teak-I: Evaluation and Verification of ANSI Y14.26M*:Booz-Allen and Hamilton, Inc.

Augenbroe, F. (2002, November 9-11). *Integration direction (Keynote).* Paper presented at the European Conference for Process and Product Modeling (ECPPM), Slovenia.

Augenbroe, G. L. M. (1993). *Combine: Final Report*.Delft, Netherlands: Delft University of Technology.

Augenbroe, G. L. M. (1995). *Combine 2: Final Report*.Delft, Netherlands: Delft University of Technology.

Bakkeren, W., A. Zarli, P. Debras, K. Schulz, and S. Korsveien. (1996). *D103a - A Model of Workflow: Specification of a Model for the Definition of Workflows in Virtual LSE Enterprises (ESPRIT 20408 - VEGA)*.

Banerjee, J., W. Kim, H. Kim, and H. Korth. (1987). *Semantics and implementation of schema evolution in object-oriented databases.* Paper presented at the Proceeding of ACM SIGMOD Annual Conference, 311-322.

Beeri, C., P. A. Bernstein, and N. Goodman. (1978). *A sophisticate's introduction to database normalization theory.* Paper presented at the International Conference on Very Large Data Bases, West Berlin, Germany, 113-124.

Benwell, G. L., P. G. Firns, and P. J. Sallis. (1991). Deriving semantic data models from structured process descriptions of reality. *Journal of Information Technology, 6*(1), 15-25.

Berners-Lee, T. (1994). *W3C: World Wide Web Consortium*. Available: http://www.w3.org/ [2004, Feb 23].

Bernstein, P. A., S. Pal, and D. Shutt. (2000). Context-based prefetch an optimization for implementing objects on relations. *VLDB Journal, 9*(3), 177-189.

Bjork, B.-C. (1989). Basic structure of a proposed building product model. *Computer-Aided Design, 21*(2), 71-78.

Bloor, M. S., and J. Owen. (1995). *Product Data Exchange*:UCL Press.

Booch, G., J. Rumbaugh, and I. Jacobson. (1999). *The Unified Modeling Language User Guide*.Reading, MA: Addison Wesley Longman, Inc.

Chakravarthy, S., S. Navathe, K. Karlapalem, and A. Tanaka. (1990). Meeting the cooperative problem solving challenge: A database-centered approach. In S. M. Deen (Ed.), *Cooperating Knowledge Based Systems*:Springer-Verlag.

Chandrasekaran, B. (1994). Functional representations: A brief historical perspective. *Applied Artificial Intelligence, 8*, 173-197.

Chen, P. (1976). The entity relationship mode - Toward a unified view of data. *TODS, 1*(1, March).

Chomsky, N. (1965). *Aspects of the Theory of Syntax*.Cambridge, MA: MIT Press.

Christiansson, P., and H. Karlsson. (1988). *CIB W74 + W78 1988 Proceeding.* Paper presented at the CIB W74 + W78, Lund, Sweden, 165-178.

CIMSteel Integration Standards Release 2. (2002). http://www.cis2.org/.

Codd, E. F. (1970). A relational model of data for large shared data banks. *CACM, 13*(6), 377-387.

Codd, E. F. (1972). Further normalization of the data base relational model. In R. Rustin (Ed.), *Data Base System* (Vol. 6, pp. 33-64). Englewood Cliffs, N. J.: Prentice-Hall.

Codd, E. F. (1979). Extending the data base relational model to capture more meaning. *ACM Transactions on Database Systems (TODS), 4*(4), 397-434.

Coplien, J. (1999). *History of Patterns http://c2.com/cgi-bin/wiki?HistoryOfPatterns* [2003, September 23].

Cover, R. (1999). *XML.org*. OASIS (Organization for the Advancement of Structured Information Systems). Available: http://www.xml.org [2004, Feb 24].

Crowley, A. (1998). *The Development & Implementation of a Product Model for Constructional Steelwork.* University of Leeds, Leeds.

Crowley, A. (1998). *The development and Implementation of a product model for constructional steelwork.* Unpublished Doctoral, University of Leeds.

Crowley, A. (2000, Jan 27, 2000). *The Logical Product Model (LPM) 5 EXPRESS Schema*. Available: http://www.cis2.org/download/lpm500.exp.

Crowley, A. J., and M. A. Ward. (1999). *CIS/2 (AP230) IDEF0*:SCI.

CSTB. (2004). *The European Research Projects (http://cic.cstb.fr/ILC/html/ecprj.htm)*.

Danner, W. F. (1997). *Developing Application Protocols (APs) using the architecture and methods of STEP (STandard for the Exchange of Product data): Fundamentals of*

the STEP methodology (NISTIR 5972). Gaithersburg, MD: National Institute of Standards and TEchnology.

Dereli, T., and H. Filiz. (2002). A note on the use of STEP for interfacing design to process planning. *Computer-Aided Design, 34*, 1075-1085.

Eastman, C., and N. Fereshetian. (1994). Information models for product design: a comparison. *Computer-Aided Design, 26*(7), 551-572.

Eastman, C. M. (1996). Managing integrity in design information flows. *Computer-Aided Design, 28*(6-7), 551-565.

Eastman, C. M. (1999). *Building Product Models: Computer Environments Supporting Design and Construction*.Boca Raton, FL: CRC Press.

Eastman, C. M. (1999). Ch 5. ISO-STEP, *Building Product Models: Computer Environments Supporting Design and Construction*.Boca Raton, FL: CRC Press.

Eastman, C. M. (1999). Ch 8. Building Framework Models, *Building Product Models: Computer Environments Supporting Design and Construction*.Boca Raton, FL: CRC Press.

Eastman, C. M. (1999). Ch 11. Modeling language Issues, *Building Product Models: Computer Environments Supporting Design and Construction*.Boca Raton, FL: CRC Press.

Eastman, C. M., S. Chase, and H. Assal. (1993). System architecture for computer integration of design and construction knowledge. *Automation in Construction, 2*(2), 95-108.

Eastman, C. M., and T. S. Jeng. (1999). A database supporing evolutionary product model development for design. *Automation in Construction, 8*(3), 305-323.

Eastman, C. M., G. Lee, and R. Sacks. (2002). *Deriving a product model from process models.* Paper presented at the ISPE/CE2002 Conference, Cranfield University, United Kingdom.

Eastman, C. M., G. Lee, and R. Sacks. (2002, June 12-14). *A new formal and analytical approach to modeling engineering project information processes.* Paper presented at the CIB W78, Aarhus, Denmark, 125-132.

Eastman, C. M., and G. V. Shirley. (1994). Management of Design Information, *Management of Design*.

Eckholm, A., and S. Fridquist. (1996). *Modeling of user organizations, buildings and spaces for the design process.* Paper presented at the Construction on the Information Highway: Proceedings of the CIB W78 Workshop, Bled, Slovenia.

Ekholm, A. (1996). A Conceptual Framework for Classification of Construction Work. *ITcon http://www.itcon.org/1996/2, 1,* 25-50.

Ekholm, A., and S. Fridquist. (1996). *Modeling of user organizations, buildings and spaces for the design process.* Paper presented at the Construction on the Information Highway: Proceedings of the CIB W78 Workshop, Bled, Slovenia.

El-Mehalawi, M., and R. A. Miller. (2001). A database system of mechanical components based on geometric and topological similarity: Part 1: representation, and Part 2: Indexing, retrieval, matching and similarity assessment. *Computer-Aided Design*, 83-94 and 95-105.

Elmasri, R., and S. Navathe. (2000). Ch.2 Database system concepts and architecture, *Fundamentals of Database Systems* (Third ed.). Reading, MA: Addison Wesley Longman, Inc.

Elmasri, R., and S. Navathe. (2000). *Fundamentals of Database Systems* (Third ed.). Reading, MA: Addison Wesley Longman, Inc.

Elmasri, R., and S. Navathe. (2004). *Fundamentals of Database Systems* (Fourth ed.). Reading, MA: Addison Wesley Longman, Inc.

EUREKA. (1987-1997). *EUREKA E!130-CIMSteel (Computer-Integrated Manufacturing For Constructional Steelwork)*. Available: http://www.eureka.be/ifs/files/ifs/jsp-bin/eureka/ifs/jsps/projectForm.jsp?enumber=130 [2003.

Feng, S. C., and E. Y. Song. (2000, November). *Information modeling of conceptual design: integrated with process planning.* Paper presented at the Symposia on Design For Manufacturability, International Mechanical Engineering Congress and Exposition 2000, Orlando, Florida.

Fenves, S. L. (2001). *A Core Product Model for Representing Design Information, NIST Internal Report 6736*:National Institute of Standards and Technology.

Fischer, M., and C. Kam. (2002). *PM4D Final Report* (143): CIFE, Stanford University.

Fowler, J. (1996, September 16-20, 1996). *Information units and views in STEP*. Paper presented at the the Interpretation Guidelines workshop, SCRA, Charleston SC.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*:Addison Wesley.

Garg, P. K., and M. Jazayeri. (1996). *Process-Centered Software Engineering Environments*.Los Alamitos, CA: IEEE Computer Society Press.

Giannini, F., M. Monti, D. Biondi, F. Bonfatti, and P. Monari. (2002). A modelling tool for the management of product data in a co-design environment. *Computer-Aided Design, 34*, 1063-1073.

Gielingh, W. (1988). *General AEC Reference Model* (ISO TC184/SC4/WG1 doc 3.2.2.1, TNO report BI-88-150).

Gielingh, W. (1988). *General AEC Reference Model (GARM).* Paper presented at the CIB W74 + W78, Lund, Sweden, 165-178.

Goldstein, B. L. M., S. J. Kemmerer, and C. H. Parks. (1998). *A Brief History of Early Product Data Exchange Standards* (NISTIR 6221 WERB). Gaithersburg, MD: National Institute of Standards and Technology (NIST).

Hammer, M., and D. McLeod. (1981). Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, *6*(3), 351-386.

Hardwick, M., K. C. Morris, D. L. Spooner, T. Randoc, and P. Denno. (2000). Lessons learned developing protocols for the industrial virtual enterprise. *Computer-Aided Design*, *32*, 159-166.

IAI.*The EXPRESS Definition Language for IFC Development*. Available: http://www.iai-international.org/iai_international/Technical_Documents/documentation/The_EXPRESS_Definition_Language_for_IFC_Development.pdf [2004.

IAI.*IAI North American Members*. Available: http://www.iai-na.org/membership/members.php.

IAI. International Alliance for Interoperability. *http://www.iai-international.org/iai_international/*.

IAI. (2000). *Industry Foundation Classes Release 2x: IFC Technical Guide*:International Alliance for Interoperability.

IAI. (2003). *Industry Foundation Classes IFC2x Edition 2*. Available: http://www.iai-international.org/iai_international/Technical_Documents/R2x2_final/index.html.

IAI. (2004). *A Short History of the IAI and the IFC Information Model*. Available: http://www.iai-international.org/iai_international/Information/History.html.

International Organization for Standardization. (1994). *ISO 10303-11:1994, Part 11: Description methods: The EXPRESS language reference manual*.

ISO JTC 1/SC 32. (2003). *ISO/IEC 9075-1:2003 Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework)* (ISO/IEC 9075-1:2003): ISO.

ISO TC 184/SC 4. (1994). *ISO 10303-1:1994 Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*:International Organization for Standardization.

ISO TC 184/SC 4. (1994). *ISO 10303-11:1994 Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual*:International Organization for Standardization.

ISO TC 184/SC 4. (1996). *ISO/DIS 10303-213:1996 Industrial automation systems and integration - Product data representation and exchange - Part 213: Application protocol: Control Process Plans For Machined Parts*.1rue de Varambe, Case Postale 56, CH-1211 Geneva, Switzerland: International Organization for Standardization.

ISO TC 184/SC 4. (1998). *ISO/CDC 10303-217 Industrial automation systems and integration - Product data representation and exchange - Part 217: Application protocol: Sheet piping*.1rue de Varambe, Case Postale 56, CH-1211 Geneva, Switzerland: International Organization for Standardization.

ISO TC 184/SC 4. (1999). *ISO 10303-14: EXPRESS-X Language Reference Manual (Working Draft)*:International Organization for Standardization.

ISO TC 184/SC 4. (1999). *ISO 10303-207:1999 Industrial automation systems and integration - Product data representation and exchange - Part 207: Application protocol: Sheet metal die planning and design*.1rue de Varambe, Case Postale 56, CH-1211 Geneva, Switzerland: International Organization for Standardization.

ISO TC 184/SC 4. (1999). *ISO 10303-225:1999 Industrial automation systems and integration - Product data representation and exchange - Part 225: Building*

elements using explicit shape representation.1rue de Varambe, Case Postale 56, CH-1211 Geneva, Switzerland: International Organization for Standardization.

ISO TC 184/SC 4. (2000). *ISO 10303-41: 2000, Integrated generic resource: Fundamentals of product description and support*:International Organization for Standardization.

ISO TC 184/SC 4. (2001). *ISO 10303-210:2001 Industrial automation systems and integration - Product data representation and exchange - Part 210: Application protocol: Electronic assembly, interconnection, and packaging design*.1rue de Varambe, Case Postale 56, CH-1211 Geneva, Switzerland: International Organization for Standardization.

ISO TC 184/SC 4. (2001). *ISO 10303-227:2001 Industrial automation systems and integration - Product data representation and exchange - Part 227: Application protocol: Plant spatial configuration*.1rue de Varambe, Case Postale 56, CH-1211 Geneva, Switzerland: International Organization for Standardization.

ISO TC 184/SC 4. (2004). *About SC4 Standards*. Available: http://www.tc184-sc4.org/About_TC184-SC4/About_SC4_Standards/.

Jablonski, S., and C. Bussler. (1996). *Workflow Management: Modeling, Concepts, Architecture and Implementation*:International Thomson Computer Press.

Jacobson, I., M. P. Jonsson, and G. Overgaard. (1992). *Object oriented software engineering: A use case driven approach*:Addison-Wesley.

Jurafsky, D., and J. H. Martin. (2000). Context-free grammar for English. In D. Jurafsky & J. H. Martin (Eds.), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (pp. 323-256). Uppper Saddle River, NJ: Prentice-Hall.

Jurrens, K. (1991). *Test Plan for Validating a Context Driven Integrated Model (CDIM) for Sheet Metal Die Design*.Gaithersburg, MD: National Institute of Standards and Technology.

Kahn, H., N. Filer, A. Williams, and N. Whitaker. (2001). A generic framework for transforming EXPRESS information models. *Computer-Aided Design, 33,* 501-510.

Karstila, K. (2001). *Precast Concrete Constructions IFC-project (PCC-IFC) Version 1.0*:unpublished working document, EuroSTEP/RTT.

Karstila, K., A. Laitakari, M. Nyholm, P. Jalonen, V. Artoma, T. Hemio, and K. Seren. (2002). *Ifc2x PCC v09 Schema in EXPRESS*:PCC-IFC project team, IAI Forum Finland.

Karstila, K., and A. Suikka. (2001). *Precast Concrete Constructions IFC - Project (PCC-IFC): Project Summary Version 1.1*:EuroSTEP/RTT.

Katranuschkov, P., J. Wix, T. Liebich, and A. Gehre. (2002). *Collected end user scenarios*:Deliverable D11, EU Project IST-2001-33022 ICCI "Innovation co-ordination, transfer and deployment through networked Co-operation in the Construction Industry".

Kogelnik, A., M. Lott, M. Brown, S. Navathe, and D. Wallace. (1998). MITOMAP: A human mitochondrial genome database. *Nucleic Acids Research, 26*(1).

Lawrence, P. (1997). *Workflow Handbook*.New York: Wiley.

McKay, A., A. de Pennington, and J. Baxter. (2001). McKay, A., de Pennington A, and J. Baxter [2001], Requirements management: a representation scheme for product specifications, Computer-Aided Design, (33, June 2001), pp. 511-520. *Computer-Aided Design, 33,* 511-520.

Nijssen, G. M., and T. A. Halpin. (1989). *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*.New York: Prentice Hall.

NIST. (1993). *FIPS Publication 183: Integration Definition of Function Modeling (IDEF0)*:National Institute of Standards and Technology.

NIST. (1993). *FIPS Publication 184: Integration Definition of Information Modeling (IDEF1X)*:National Institute of Standards and Technology.

NIST. (2002). What is STEP. *http://cic.nist.gov/plantstep/stepinfo/step_def.htm*.

OMG. (2003). *Object Management Group (OMG) http://www.omg.org* [2003, September 23].

Osborne, L. N., and M. Nakamura. (2000). *Systems Analysis for Librarians and Information Professionals* (2nd ed.). Englewood, Colorado: Libraries Unlimited, Inc.

P. Lawrence (Ed.). (1997). *Workflow Handbook*.New York: Wiley.

Pahl, G., and W. Bietz. (1998). *Engineering Design: A Systematic Approach*:Springer-Verlag.

Palmer, M. E., and K. Reed. (1990). *3D Piping IGES Application Protocol version 1.0*:National Institute of Standards and Technology, Interagency Report 4420.

Petri, C. A. (1962). *Fundamentals of a theory of asynchronous information flow.* Paper presented at the IFIP Congress, North Holland, 386-390.

Renssen, I. A. v. (1997). *ISO CD 10303 Guide on STEPlib: Guide for the creation and maintenance of Standard Data for Process Plants (Ver. 1.8)* (ISO TC184/SC4/WG3/N424). The Hague, The Netherlands: Shell International Oil Products B.V.

Ronneblad, A. (2003). *Product models for concrete structures: standards, applications and implementations.* Unpublished Licentiate Thesis, Lulea University of Technology, Sweden.

Rosenberg, D., and K. Scott. (1999). Robustness Analysis, *Use Case Driven Object Modeling with UML: A Practical Approach* (pp. 61-79). Reading, MA: Addison Wesely Longman, Inc.

Rosenberg, D., and K. Scott. (1999). *Use Case Driven Object Modeling with UML: A Practical Approach*.Reading, MA: Addison Wesely Longman, Inc.

Rusinkiewicz, M., and A. Sheth. (1995). Specification and execution of transactional workflows. In W. Kim (Ed.), *Modern Database Systems: the Object Model, Interoperability, and Beyond* (pp. 592-620): ACM Press.

Sacks, R., C. M. Eastman, and G. Lee. (2004). Process model perspectives on management and engineering procedures in the North American Precast/Prestressed Concrete Industry. *the ASCE Journal of Construction Engineering and Management, 130*(2), 206-215.

Schenk, D. A., and P. R. Wilson. (1994). *Information Modeling the EXPRESS Way*.NY:Oxford U. Press.

Shipman, D. W. (1981). The functional data model and the data languages DAPLEX. *ACM Transactions on Database Systems (TODS), 6*(1), 140-173.

Smith, G. L. (2002). Utilization of STEP AP 210 at the Boeing Company. *Computer-Aided Design, 34*(14), 1055-1062.

Smith, J. M., and D. C. P. Smith. (1977). Database abstractions: aggregation and generalization. *ACM Transactions on Database Systems (TODS), 2*(2), 105-133.

Smith, J. M., and D. C. P. Smith. (1997). Database abstractions: aggregation. *Communications of the ACM, 20*(6), 405-413.

Smith, N., and D. Wilson. (1979). *Modern linguistics: The results of Chomsky's revolution*:Penguin.

Spooner, D. L., and M. Hardwick. (1997). Using views for product data exchange. *IEEE Computer Graphics and Applications, 17*, 58-65.

Stouffs, R., R. Krishnamurti, and C. M. Eastman. (1996, 16-18 September 1996). *A formal structure for non-equivalent solid representations.* Paper presented at the

International Federation for Information Processing (IFIP) WG 5.2 Workshop on Knowledge Intensive CAD II, Pittsburgh, Pa, 269-289.

Szykman, S., S. Fenves, W. Keirouz, and S. Shooter. (2001). A foundation for interoperability in next-generation product development systems. *Computer-Aided Design, 33*, 545-559.

Tari, Z., J. Stokes, and S. Spaccapietra. (1997). Object normal forms and dependency constraints for object oriented schemata. *ACM Transactions on Database Systems, 22*(4), 513-569.

Tolman, F. P., and P. Poyet. (1995). *The ATLAS models.* Paper presented at the Product and Process Modelling in the Building Industry, Rotterdam.

Turner, J. A. (1988). *AEC Building Systems Model* (ISO TC184/SC4/WG1 doc 3.2.2.4).

Turner, J. A. (1988). *A systems approach to the conceptual modeling of buildings.* Paper presented at the CIB W74 + W78, Lund, Sweden, 179-194.

VTT. (2004). *Extension Projects for IFC (http://ce.vtt.fi/iaiIFCprojects/).*

VTT Building and Transport. (2002). Construction IT Glossaries. *http://cic.vtt.fi/links/glossary.html*.

Weikum, G. (1991). Principles and realization strategies of multilevel transaction management. *ACM Transaction on Database Systems, 16*(1), 132-180.

WFMC. (1999). *Workflow Management Coalition Terminology & Glossary* (WFMC-TC-1011). Hampshire, UK: WFMC.

Wilson, I., S. Harvey, R. Vankeisbelck, and A. S. Kazi. (2001). Enabling the construction virtual enterprise: The OSMOS apprach. *ITcon, 6*(Information and Communication technology advances in the European construction industry), 83-110.

Wilson, I., S. Harvey, R. Vankeisbelck, and A. S. Kazi. (2001). Enabling the construction virtual enterprise: The OSMOS apprach. *ITcon, 6*, 83-110.

Wix, J., and P. katranuschkov. (2002, June). *Defining the matrix of communication processes in the AEC/FM industry: Current developments and gap analysis.* Paper presented at the CIB w78 Conference, Aahus School of Architecture, Denmark.

Wix, J., and T. Liebich. (2000). *Information flow scenario*:Deliverable D4, EU Project IST-1999-11508 ISTforCE, "Intelligent Services and Tools for Concurrent Engineering".

Wix, J., and T. Liebich. (2000). *Information flow scenario: Deliverable D4, EU Project IST-1999-11508 IST-forCE(Intelligent Services and Tools for Concurrent Engineering)*.

Wyke, R. A., and A. Watt. (2002). Ch. 10 Bringing the Parts Together, *XML Schema Essentials* (pp. 305-346). New York, NY: Jony Wiley & Sons, Inc.

Yang, D., S. You, F. Wang, C. M. Eastman, and J. Lee.*CIS/2 @ Georgia Tech.* Available: http://www.arch.gatech.edu/~aisc/.

You, S.-J. (2003). File merging for avoiding import-time data loss in a file-based STEP implementation. *Qualifying paper (unpublished), College of Architecture, Georgia Institute of Technology, Atlanta GA, USA.*

You, S.-J., D. Yang, and C. M. Eastman. (2004, May 2-7, 2004). *Relational DB Implementation of STEP based product model.* Paper presented at the CIB World Building Congress 2004, Toronto, Ontario, Canada.

VITA

Ghang Lee received a bachelor's degree in architectural engineering and a master's degree in architectural design both from Korea University respectively in 1993 and 1995. After graduation, he worked at Kumho, a large-scale construction company in Korea, for about five and a half years as an architectural designer, researcher, software developer, and later as an assistant manager. While working at Kumho, he and his team received a Medal of Merit from the President of Korea in quality management of apartment complex design and construction for research-oriented apartment complex design.

He began his Ph.D. study at Georgia Tech in 2000. During his stay at Georgia Tech, he was involved in various advanced CAD system development projects and data modeling (e.g., ISO-STEP, IFC, CIS/2) projects and taught a graduate-level course on parametric modeling. Through the various research projects, he worked with a wide range of organizations and companies such as the North American Precast Concrete Software Consortium (PCSC), Teka Oy, the Sustainable Facilities & Infrastructure (SFI) group at Georgia Tech Research Institute (GTRI), the Design in the Classroom (DITC) group at College of Computing, and the Southern Staircase.

He has published twenty refereed journal and conference papers and ten project reports since 1996. One of them was a keynote at the European Conference on Process and Product Modeling (ECPPM) in 2002. He has served as a session chair at international conferences such as CIBw78 and ECPPM and as a conference organizer ("BFC05: the first conference on the Future of the AEC Industry"). He developed two licensed software applications and several interactive websites.