Deriving Process Specifications from Augmented Data Models

Götz Botterweck
Institute for IS Research
University of Koblenz-Landau
Universitaetsstr. 1
56070 Koblenz, Germany

E-mail botterwe@uni-koblenz.de
Phone +49 261 2872531
Fax +49 261 2871002531

Carlo Simon
Institute for Management
University of Koblenz-Landau
Universitaetsstr. 1
56070 Koblenz, Germany

E-mail simon@uni-koblenz.de
Phone +49 261 2872861
Fax +49 261 2872851

Abstract

This paper presents an approach which enables a system analyst to derive process models from data models. To accomplish this, existing data models are augmented with additional information regarding the intended process goal and the dependencies which have to be fulfilled.
After giving an introductory example and proposing a notation for the augmentations, a simple algorithm is sketched. This derives a process model consisting of all steps necessary to fulfil the specified goal and the related dependencies.

1. Introduction

Following a separation of concerns, it is common in software engineering (SE) to describe separate aspects of the represented domain in separate models. Typical examples are structure-oriented models and models which describe the dynamic behaviour of a system.

As these models represent different views on the *same* object, it is only natural that they are related to each other and partially overlap in the concepts they describe.

Modelling methods usually provide hints and heuristics about the order in which the various models should be created and refined. When following these suggestions, some concepts in one model can often be derived from other models. These derivations are possible between many types of models and in many directions. However, surprisingly little work has been conducted in the area of deriving process models from data structures.

We will address this by presenting a corresponding approach. Section 2 sets the stage by describing common use of distinct models in SE. Section 3 gives an overview of related work, especially process-driven approaches. Section 4 presents our approach, starting with an introductory example, and then discussing how to derive the process descriptions from augmented data models. Two more advanced examples are included. Section 5 closes by assessing some features of the presented approach and describing opportunities for future research.

2. Distinct Models in Software Engineering

To describe all relevant aspects of a modelled domain, state-of-the-art modelling languages usually offer several model types – and are therefore actually *sets* of interlinked languages. One prominent example is the UML [1] which provides, for instance, scenario-oriented models (use cases), structure-oriented models (e.g. class diagrams) and behaviour-oriented models (e.g. statechart and activity diagrams). Other object-oriented methodologies like OPEN [2] suggest similar concepts.

The idea of such approaches is not to consider the different modelled perspectives on software in isolation but to combine these views. So, static aspects like the data structure are linked with dynamic aspects describing the methods of the classes and their interplay. As an example, consider a class in a class diagram describing the inner structure and relationships to other classes, and related statechart diagrams describing its behaviour. These connections are often formally defined in a meta model; in UML for instance the meta model states that each *ModelElement* can have one or more *StateMachine* describing its behaviour [3].

A similar but slightly different way of thinking is the concept of having just *one* model with the "different descriptions (static, dynamic, use case) [being] different

views of aspects of the same entity" as Henderson-Sellers puts it [2].

In the discussion of an appropriate way of eliciting and describing the suggested models, almost all modelling methodologies give hints about the order in which these models should be created. For example, UML suggests to initially develop use cases, i.e. pictorial descriptions of the interaction between the users, the application and further aspects of the environment relevant for the execution of the software.

Software development processes help to progress from relatively informal descriptions of the problem to precise and executable software. Within each single refinement step, previous results are taken as input and are further developed. Consequently, there is a strong relationship between the elements of the designed models.

As two related models describe some overlapping aspects (cf. Figure 1) some of the information in one model can be derived from the other and then be augmented by aspects only described in the second model.

This derivation can be done in both directions and the question of which model should be created first depends on the context and the perspective of the system analyst.

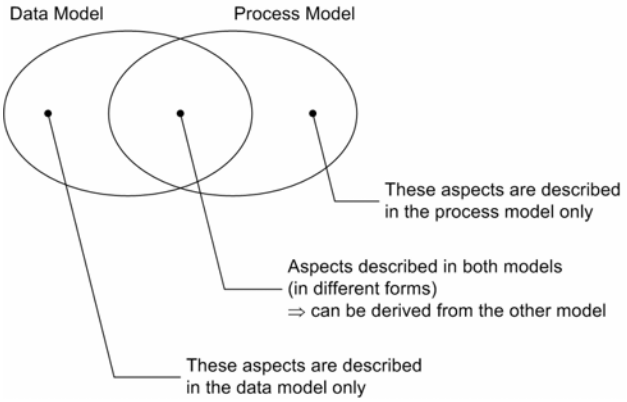

Figure 1. Overlapping aspects in different model types – here data models and process models.

In other cases with an even larger overlap, almost all the information can be directly derived from the other model, e.g. for sequence diagrams and collaboration diagrams illustrating aspects of the same interaction in different presentation forms.

When deriving and refining models, we have to consider the goal of *traceability*, which is the ability to reconstruct the path the requirements haven taken through the different representations (narrative descriptions, formal models, source code) during the different phases of software engineering.

Another challenge is the goal of *consistency* of the whole set of models. If we use overlapping models to describe the same object from different perspectives, we inevitably create redundancy, and consequently run the risk of inconsistencies between these models [4, 5]. As an example, consider a use case describing a scenario in one way and related class and statechart diagrams describing the same scenario in another, conflicting way [6]. This problem is intensified even further by the fact that the semantics of common modelling language is often not defined in an unambiguous formal way [7].

It should be noted that most modelling methodologies describe the *sequence* in which models should be created only as a logical ordering of the models, but promote an iterative approach (or an approach where the different models are refined side-by-side) as a process model [8]. This is similar to overall software engineering approaches describing sequences of phases to provide a logical structuring (e.g. analysis, design, implementation, transition), but then promoting iterative process models for the practical approach [9].

As can be seen in the following section, approaches which begin with a process view on the problem have become quite popular in the past years.

3. State of the Art

The development of information systems is typically process-driven. We start by understanding the underlying business processes and capturing them in a narrative way, a graphical or formal mathematical notation. The degree of formalization depends on the chosen notation: Event-driven Process Chains (EPC) [10] are a semi-formal notation based on Petri nets [11, 12] but without a defined semantics [13]. Other approaches like Workflow-nets [14], also a Petri-net based approach, describe business processes in such a way that the resulting models can be executed in a Workflow Management Engine [15].

On the basis of such a precise specification of the system to be modelled, data and resources must be associated with each single task in order to complete the specification for execution. Figure 2 visualizes this development process.

Earlier approaches to software development like Structured Design [16] are also process- driven. In this methodology, however, the lack of separation of data and control causes problems when it comes to interpreting such models.

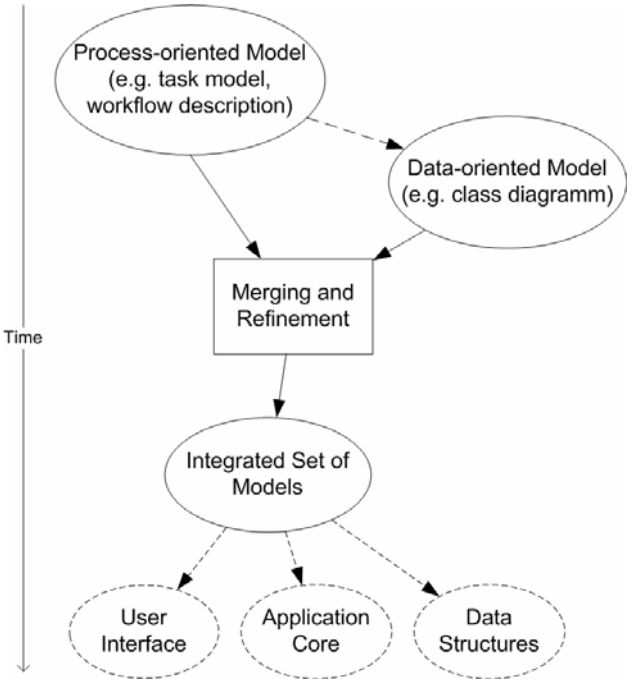

Figure 2. Process-driven approaches

Process-driven approaches to software development are quite often chosen by development teams because of their ability to visualize the strong relationship between the business processes to be supported and the way the information system is executed. Moreover, experience shows that users who are involved in the software development process feel especially comfortable with process visualization techniques.

Process-centred approaches also typically provide analyzing techniques which give clear insights into the modelled processes. Two examples for this are ConcurTaskTrees (CTT) [17] which allow to determine all tasks that can possibly be enabled at the same time, or a Logic of Actions (LoA) [18, 19] which proves process implementations against process specifications. Finally, we want to mention Process Algebra [20] as a formalism to specify and analyze processes.

Temporal Logic [21], on the other hand, concentrates more on states and whether specific states are reachable in the future or not. As such, Temporal Logic is not focussed on the actual processes but is more a link between process-oriented and data-oriented approaches.

While a lot of research has been conducted on how to find appropriate process models and how to analyze them when developing new software, surprisingly little has been done on deriving (new) process models from given applications or their specifications.

One example of such an approach is the mining of workflows from data collected during the monitoring of workflow management systems [22]. An application is in the field of software reengineering, especially if the underlying formal workflow model is lost. This approach allows the model to be reconstructed and the application to be reengineered.

In the following, we consider another approach in which we use existing data models to rapidly derive process models. The data models are augmented with information concerning the sequence of access to the data. Subqueries and sequences of forms can be derived from this information in a structured way.

The resulting process models can be used for both the analysis of the implemented workflows with respect to existing business processes, and the generation of user front-end descriptions.

As we are discussing the derivation of one model from another, we have to mention the generation of user interface descriptions from these declarative models.

A first, rather primitive mechanism in this sense is the functionality in common data-oriented tools which allow the user to create a simple ad-hoc user interface from a data model. In practice, this is not usually described as a distinct model on its own but is available in the form of the schema of the currently used database (cf. Figure 3).

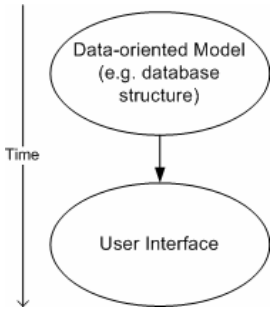

Figure 3. From data structures to user interfaces

These simple mechanisms do not include dedicated process or task models and therefore can only create schematic user interfaces. The interfaces provide support for standard operations like selecting and updating existing records or creating new ones. Examples of such tools are wizards in common database software, which allow the user to generate data manipulation forms based on the data structure of the underlying database (cf. Figure 4).

More advanced research projects aim for the development or even the automatic generation of user interfaces based on declarative models [23-27]. Tools developed in the context of such projects are therefore often called Model-Based (User) Interface Development Environments (MB-IDEs). Detailed overviews of such approaches can be found in [28-30].

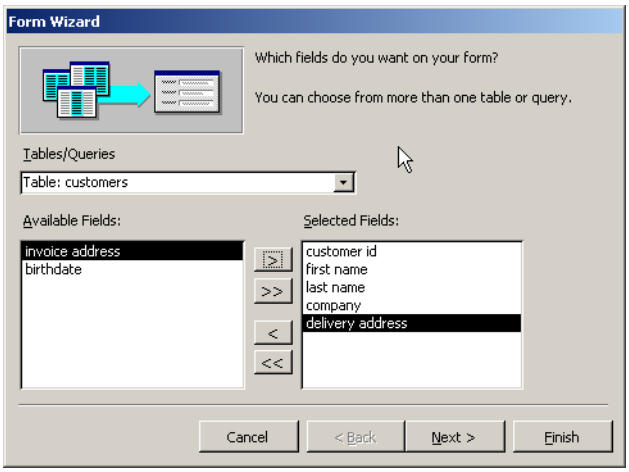

Figure 4. Tool support to generate a simple UI from a database structure, here Microsoft Access.

Almost all of these approaches use domain models and/or task models, as the foundation for the generation of the user interface (cf. Figure 5). Some also include a user model describing constraints and requirements given by properties of the envisioned user or a dialogue model. This corresponds to the steps described in the task model, but is more detailed and more oriented on interactions.

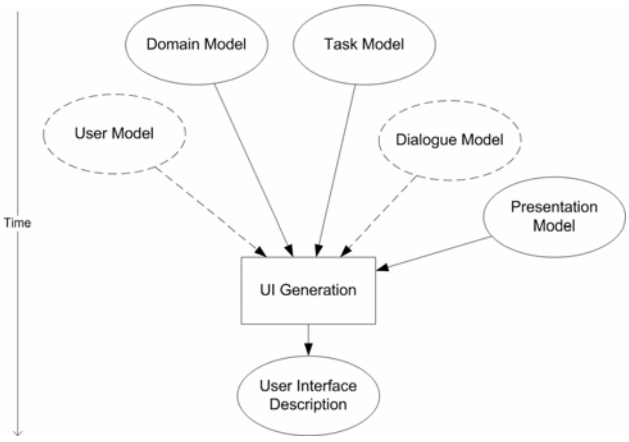

Figure 5. Model-based development of user interfaces

From these numerous model-based approaches, we will focus here on the above-mentioned, task-oriented CTT [17], since corresponding mechanisms can be employed to take further advantage of process models which have been elicited by our approach by deriving user interface descriptions from them. In particular it is worth mentioning TERESA [31], a tool which – besides being a modelling tool for task descriptions in CTT – allows us to generate user interfaces from these task descriptions. In this context it explicitly addresses the challenge of developing multiple front-ends (e.g. GUI, PDA, mobile phone, speech). The related approach suggests creating refined task models for each device – a step which has to be performed by manual augmentations and refinements by the designer – and then generating the different user interface descriptions from them.

4. Deriving Tasks from Augmented Data Models

4.1 Example: Cinema Ticketing

As an introductory example we will have a look at a fictitious cinema ticketing application. To keep this first example simple, we will assume that a ticket allows you to see a certain show (i.e. a certain movie in a certain cinema at a certain performance time), but does not include reservation of a particular seat. Figure 6 shows the related UML data model including augmentations (dashed lines and note symbols) which are added during the analysis process.

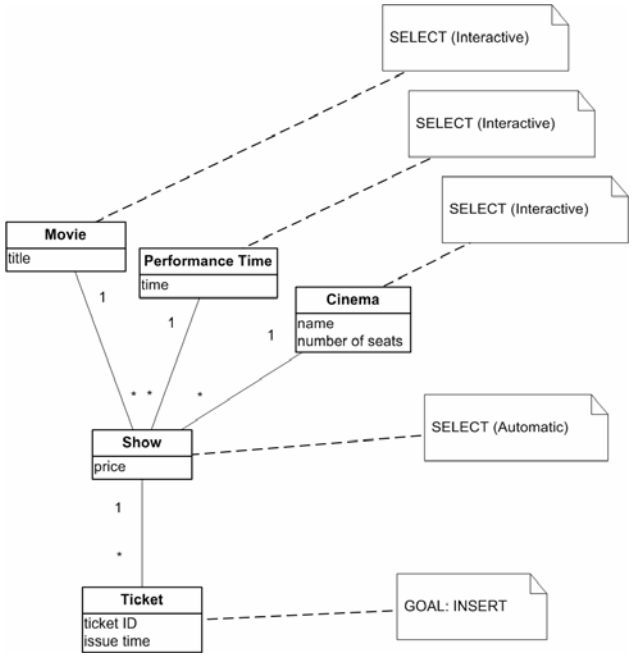

Figure 6. Cinema ticketing, UML data model and augmentations (dashed lines and note symbols)

We assume that the non augmented data model is given by the current cinema management system which functions quite well in principle. We will now review this data model in order to identify new processes which could be supported by the information available. For in-

stance, instead of selling cinema tickets at a ticket window, we want to investigate the possibility of selling them at a ticket machine or even by mobile devices. For this it is necessary to conduct a process analysis.

From a data perspective, buying a ticket means inserting a new record or object into the *Ticket* class. Here we have to determine all object attributes and all related objects: each ticket is related to exactly one *Show* which is related to exactly one *Movie* at a specific *Performance Time* and in a specific *Cinema*. A customer has to select these parameters, but is not allowed to insert more records (e.g. to add an additional Movie).

If we augment the *Ticket* class by the information that our primary goal is to buy a ticket (i.e. insert a corresponding record), then this is achieved by a process which we call level 0 process. This process can be divided into the actual insertion and the preparation of all parameters necessary for it (level 1). On level 2 we have the interactive selections, which are subdivided into further subprocesses on level 3 responsible for the selection of *Movie*, *Performance Time*, and *Cinema*, and the resulting selection of the chosen *Show*. The actual insert operation can be subdivided into three steps (level 2): an insert confirmation, writing a new data object to the database and a feedback concerning a successful operation, including a code number (e.g. printed as a bar code) which enables the entrance into the show room.

It should be noted that this *three-step insertion* (asking for confirmation, actual database insert, and feedback) is not a direct consequence of the augmented data model, but rather a kind of subprocess design pattern we use for all similar insertion processes.

Figure 7 visualizes the resulting overall process for the cinema ticketing example as a ConcurTaskTrees (CTT) diagram. From the information given by the augmented data model (Figure 6) we derive a tree of tasks. Its root results from the *goal* of the augmented data model (Buy Ticket). All other nodes describe subtasks which have to be performed to reach that *goal*.

In CTT, the different node symbols indicate whether a particular subtask is carried out by the technical system alone (computer icon), the user alone (portrait icon, not used here), by the user interacting with the system (person with computer) or is aggregated from a mixture of subtasks of different types (cloud icon).

The subtasks are connected by operators which describe temporal relationships between them. Some examples are:

- |=| Independent Concurrency – The tasks connected with this operator can be performed in any order.
- [] Choice – Any of the tasks can be performed.
- >> Enabling – One task enables the execution of the next one when it finishes.
- []>> Enabling with Information Passing – One task provides information for the next task.

Consider, for instance, the two child nodes of the root: to buy a ticket you have to specify all necessary parameters (left subtree) and *then* (temporal operator []>>) insert the new *Ticket* record (right subtree).

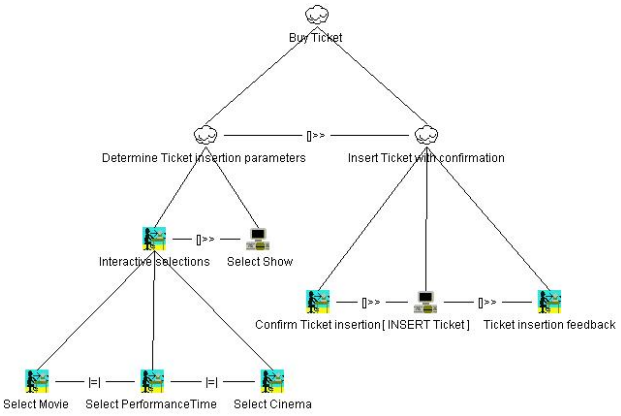

Figure 7. Cinema ticketing, resulting CTT task model

For more details concerning the CTT notation we refer to [17].

4.2 Overview of Our Approach

We use a UML data model as the input for our approach (cf. Figure 8). This data model is initially annotated manually with the intended goal ❶ (e.g. the *Ticket* class is annotated with the goal *INSERT*).

Suggestions for further annotations can then be automatically derived ❷ by considering constraints given in the data model (e.g. to insert a *Ticket* record we have to fulfil all relationship and cardinality constraints. Hence, we have to provide related *Movie*, *Performance Time* and *Cinema* records which probably have to be selected or inserted. The application designer decides which of these operations – select, insert, or both – are available to the user.

As an option, the analyst may also manually add augmentations ❸ which may lead to additional implications ❹. Further iterations might be required to complete the model.

This leads to a data model annotated with a goal and additional augmentations which can be considered as subgoals. This final model has to be transformed into a process model in CTT notation ❺.

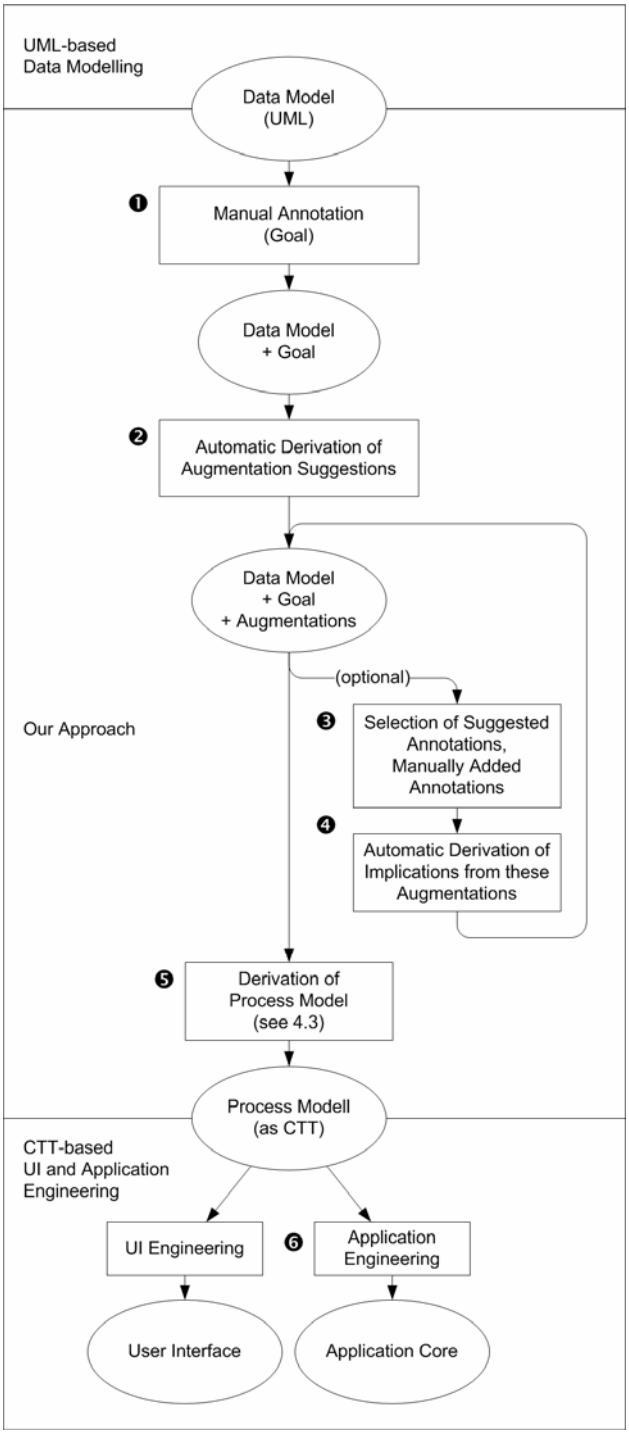

Figure 8. Overview of our approach

This transformation (for details see the following section 4.3) results in the final output of our approach: a process-oriented model in CTT notation, which can then be employed as a foundation for the engineering of an application and its user interface.

4.3 Task Generation from Augmented Data Models

Process models are derived from data models by augmenting them with intended operations on the represented data. In principle, four different operations on the data entities must be describable: *insert*, *delete*, *update* and finally *select* in the case of pure information retrieval. These operations can be applied to both types of information represented in classes and the relationships among them. In the case of an *update*, we imagine that we also have finer granular operations on an attribute's level. We do not consider them in more detail here, since they are of no further relevance in this context.

In our initial example, we furthermore observe that we can differentiate *levels* of closeness of such an operation to a primary goal defined in terms of modify or query operations. In the resulting process model, these closeness levels will be translated into hierarchy levels in a tree-shaped process structure, with the primary goal represented by the root of the tree.

In the example, the insertion of an instance of a specific class can be seen as a primary goal. However, this object can only exist in conjunction with others which have to be selected first. Then these selections are subordinate to the primary goal and therefore on the next level.

Both augmented level and augmented operations cannot be derived automatically from the data model in all cases since they represent world knowledge. In our initial example, we have to know that visitors of a cinema are not allowed to modify the program (only select operations are allowed on *Show*, *Movie*, *Performance Time* and *Cinema*). They are only allowed to buy tickets (i.e. insert entities to *Ticket*).

Although we can derive the sequence in which operations must be conducted from cardinality information of the data model in many cases (especially for 1:1 and 1:n relationships), this is not possible in general. For example, the data model defines (by cardinality) that each ticket requires, among other related objects, an associated *Show*. Consequently, a select or an insert operation on objects of this class must also be conducted when a ticket is inserted. In relationships with n:m cardinality, knowledge of the problem domain is required to determine the level correctly. Therefore, both level and intended operation must be augmented by a domain expert. A tool could suggest specific levels.

These considerations lead to the simple meta model of data models shown in Figure 9. In this model, we left away concepts like generalisation-specialisation and aggregation as this gives us more space for considering two additional examples in the following. There we do not make use of these concepts, but they can be integrated in a straightforward way.

A meta model is needed for our target modelling language in order to derive a task model from our augmentations. We chose to generate CTT models already mentioned in the third section. A meta model for CTTs can be found at [32].

Now we are able to sketch an algorithm for the generation of the process structure. As input, we assume a given augmented data model represented in a data structure in accordance with the meta model of Figure 9. As output, we generate a CTT model in accordance with the meta model of CTT cited above. For each augmented class and relationship, both operation and level information are assumed to be explicitly given.

The fulfilment of the *goal* operations depends on the successful execution of all operations on level 1 which are linked with the considered class or relationship. Therefore, the implementation of these operations is considered as a sub goal. Our algorithm is recursively applied to level 1 elements and to all following levels. For these, a hierarchy is generated in the CTT model.

Afterwards – on the hierarchy of the CTT model which handles the parallel execution of the level 0 operations – a generation pattern is applied to each of the specific operation types (insert, delete, update, or select). We already described the three-step insertion in section 4.1 as an example for such a pattern. A pattern for a combined select-or-insert operation is discussed in the following section.

As a final remark, it is possible that one and the same class or relationship is augmented with several operations at different levels. Although this causes no problems for our algorithm, the task of reading and understanding data models augmented this way can be confusing.

4.4 Example: Purchase Order Processing

We will now consider a second example which deals with a purchase order processing application. Imagine an agent of a retailing company answering calls and taking customer orders. For simplification we will assume each purchase order contains only one product (and not a list of purchase order items). Hence, each *PurchaseOrder* is related to exactly one *Customer* and exactly one *Product* (cf. Figure 10).

From a data perspective, creating a purchase order means inserting a *PurchaseOrder* record which is our *goal* for the ongoing augmentation. Hence, the insertion of a purchase order will be our level 0 process. By analysing the outgoing relationships and their cardinalities, we can conclude that we must have exactly one *Customer* and exactly one *Product* to be associated to the newly-created *PurchaseOrder*.

Again the information whether these records already exist (*select*) or have to be created (*insert*) in the process is world knowledge and, hence, has to be specified as an

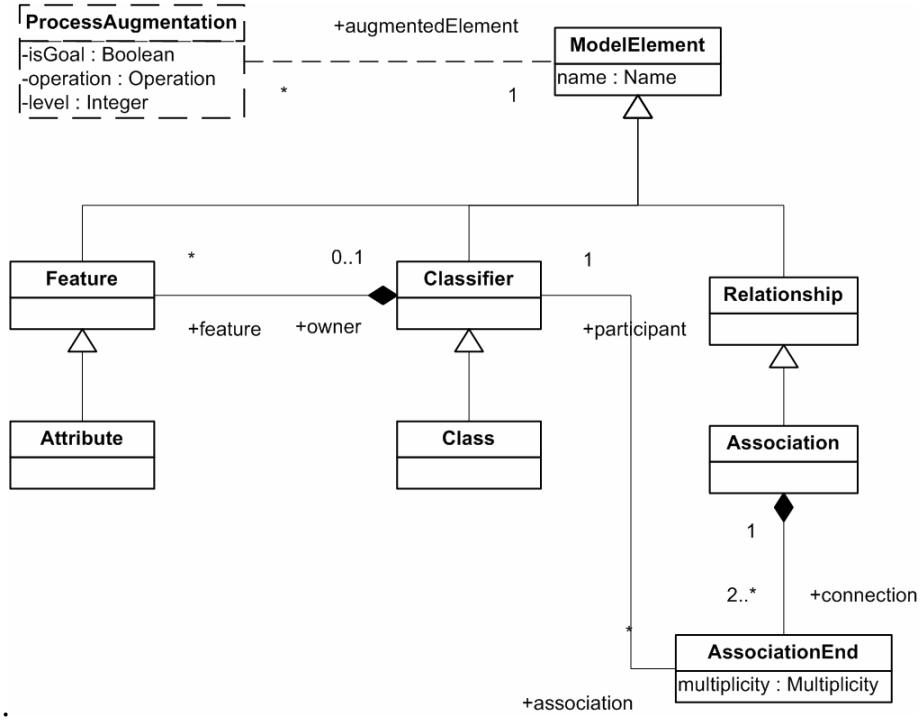

Figure 9. Excerpt from the UML meta model with augmentation extensions (dashed)

augmentation by the designer.

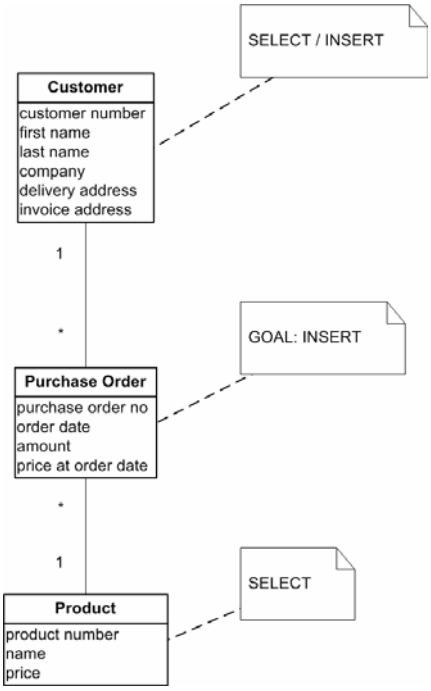

Figure 10. Purchase order processing, UML data model and augmentations (dashed lines and note symbols)

Although it might be desirable in real life that customer requests create new business ideas, we will – in the small world of our simplified example – assume that customer calls never trigger the creation of new products. Consequently, the *Product* class is augmented with a *select* label and can be handled similarly to the previous example.

It might, however, happen that a *new* customer is contacting us, i.e. someone for whom we have no *Customer* record yet. As we already have a small basis of existing records and the calling customer might be in there, we augment the *Customer* class with a *Select/Insert* Label. This will be transformed into a process subtree describing the steps necessary for an attempted select (when the agent is searching for matching records) and an optional insert if the required record is not found.

These two subprocesses (*select* of a *Product*) and (*Select/Insert* of a *Customer*) can be performed independently as there are no dependencies between them, i.e. relationships between the affected classes in the data model. Together they form a subprocess which provides the parameters necessary for the *PurchaseOrder* insertion.

The insertion itself, as in the previous example, is expressed in the CTT task tree (c.f. Figure 11) as a combination of confirmation (showing the record to-be-inserted), an insert transaction (which might be skipped if the user does not confirm it) and a feedback screen showing the results to the user.

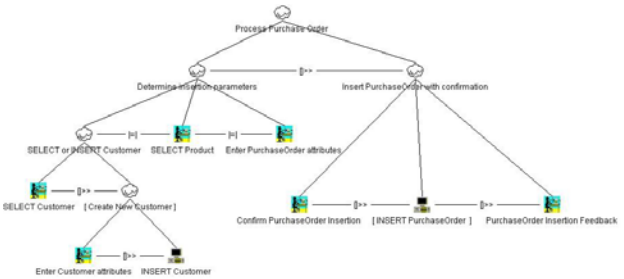

Figure 11. Purchase order processing, resulting CTT task model

4.5 Example: Product Catalog

With the following final example, we want to focus on the discussion of whether the augmentations should be done on the schema (class) level or on the instance (object) level.

Imagine a simple *ProductCatalog* consisting of *ProductCategories* which themselves contain *Products*. Product categories are hierarchically organized, so a *ProductCategory* can have one or zero parent categories (cf. Figure 12).

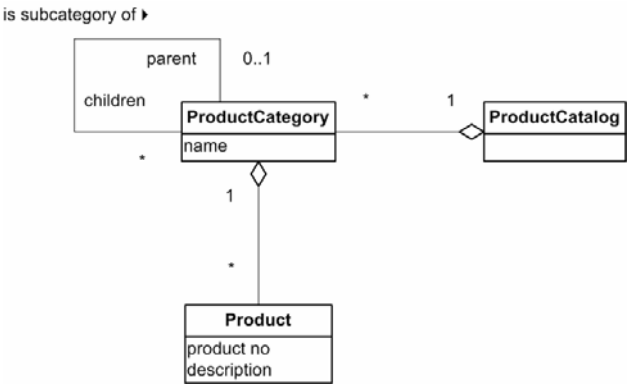

Figure 12. Product catalog, data model (class diagram)

Now imagine an existing catalog content management application which we want to extend by a new process that supports the insertion of new subcategories. We cannot really express this on schema level (with an augmented class diagram) since we are talking about two instances of *ProductCategory* here: the existing super category and the to-be-inserted subcategory. This can be expressed by an object diagram (Figure 13) fulfilling the

static structure laid out by the related class diagram shown before.

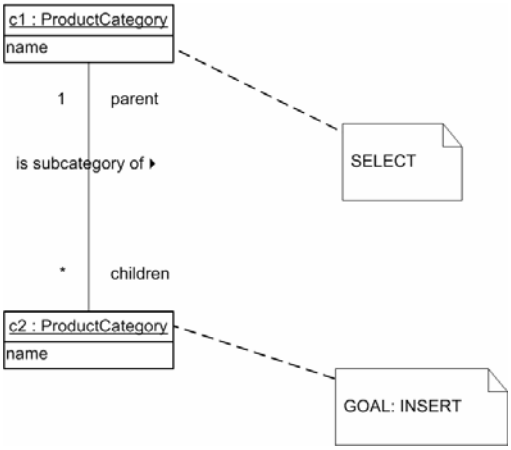

Figure 13. Product catalog, data model (object diagram with augmentations)

Our goal is to insert a new *ProductCategory* object (c2). This operation becomes our level 0 process. Before we can do the already known three-step insertion, we have to determine all required parameters. These are the attributes of the new object and all related objects. In this example, the user has to interactively select a *Product Category* (c1) which then will be associated as a super category to the new product category c2.

Figure 14 shows the resulting task model, which largely corresponds to the structure already discussed in the preceding examples.

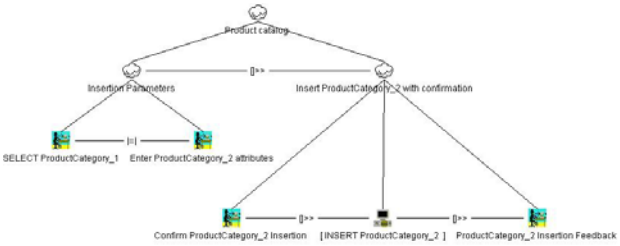

Figure 14. Product catalog, resulting CTT task model

5. Conclusion

The need to find process specifications exists for both iterative software engineering and reengineering. We demonstrated in our paper how to rapidly find such specifications when a data model already exists.

Our approach is based on augmenting classes and relationships of data models by primary goals (like *insertion* or *deletion)* and by subgoals needed to fulfil the primary goal. These goals can be evaluated concerning their closeness to the primary goal by introducing a level concept. An algorithm generates a process model describing the sequence in which the (sub-) goals have to be fulfilled. Especially if we systematically annotate classes and relationships of our data model by possible goal operations, we can evaluate applications concerning their completeness with respect to the set of processes that could be implemented.

The generated process models can be used for verification and for generating application source code or a user interface description.

Our approach is novel in the sense that we describe how to derive process models from data models while the reverse is typically investigated in software engineering. This allows us to reveal formerly hidden processes by only relying on a given data model.

Based on these results we see several opportunities for future work. We plan to further improve our approach on the attribute level of data models, especially exploring the use of attributes for the generation of finer granular processes which, beside the pure process structure, also provide information for the initialisation of new entities.

We intend to further refine and formalise the presented notation by defining a more detailed meta model. Here we see an important task in integrating our augmentations with existing notations for data models.

Besides being desirable in itself, this would simplify the next step: the implementation of an application which provides tool support for the presented approach: the interactive augmentation of existing data models and the generation of corresponding process models (e.g. in the mentioned CTT notation). They could then be processed with tools focusing on process models, for instance to generate user interface descriptions from them.

An open question is the problem of having processes which are operated by several users. Here, our process model must allow the delivery of intermediary result to others. Further annotations may have to be introduced to solve this.

References

[1] Object Management Group, "OMG Unified Modeling Language 1.5," 2003.

[2] B. Henderson-Sellers, "OPEN: Object-oriented Process, Environment and Notation - The first full lifecycle, third generation OO method," in Handbook of Object Technology, S. Zamir, Ed. Boca Raton, Florida, United States: CRC Press, 1997.

[3] Object Management Group, "State Machines," in OMG Unified Modeling Language 1.5, 2003, pp. 2/140-2/169.

[4] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen, "A methodology for specifying and analyzing consistency of object-oriented behavioral models," presented at Proceedings of the 8th European software engineering conference, Vienna, Austria, 2001.
[5] C. Pons, R. Giandini, G. Baum, J. L. Garbi, and P. Mercado, "Specification and checking of dependency relations between UML models," in UML and the unified process. Hershey, PA, USA: Idea Group Publishing, 2003, pp. 237-253.
[6] M. Glinz, "A Lightweight Approach to Consistency of Scenarios and Class Models," presented at Proceedings of the 4th International Conference on Requirements Engineering (ICRE'00), 2000.
[7] M. Prasse, "Entwicklung und Formalisierung eines objektorientierten Sprachmodells als Grundlage für MEMO-OML." Koblenz, Germany: Universität Koblenz-Landau, 2002.
[8] P. Kruchten, The Rational Unified Process: An Introduction, 3rd ed: Addison Wesley, 2003.
[9] J. Garland and R. Anthony, "Overview of Iterative Development," in Large-Scale Software Architecture. Chichester, England: John Wiley and Sons, 2003, pp. 39-48.
[10] A. W. Scheer, "Business Process Engineering," in ARIS-Navigator for Reference Models for Industrial Enterprises. Berlin, Germany: Springer, 1994.
[11] W. Reisig and G. Rozenberg, Lectures on Petri Nets I: Basic Model / II: Applications (vol. 1491/1492 of LNCS). Berlin, Germany: Springer, 1998.
[12] T. Murata, "Petri Nets: Properties, Analysis, and Applications.," in Proc. of the IEEE, vol. 77(4), 1989, pp. 541-580.
[13] E. Kindler, "On the Semantics of EPCs: A Framework for Resolving the Vicious Circle," presented at Business Process Management (BPM 2004) (vol. 3080 of LNCS), Potsdam, Germany, 2004.
[14] W. M. P. v. d. Aalst, "The Application of Petri Nets to Workflow Management," The Journal of Circuits, Systems and Computers, 1998.
[15] Workflow Management Coalition, "Terminology & Glossary (issue 2.0)," 1996.
[16] E. Yourdan and L. Constantine, Structured Design: Prentice-Hall, 1979.
[17] F. Paternò, Model-Based Design and Evaluation of Interactive Applications. Berlin: Springer, 2000.
[18] C. Simon, "A Logic of Actions and Its Application to the Development of Programmable Controllers." Koblenz: University of Koblenz-Landau, 2001.
[19] C. Simon, "A Logic of Actions to Specify and Verify Process Requirements," in The Seventh Australian Workshop on Requirements Engineering (AWRE'2002). Melbourne, Australia, 2002.
[20] W. Fokkink, "Introduction to Process Algebra," Springer, 2002.
[21] Y. Shoham, "Temporal Logics in AI: Semantical and Ontological Considerations," Artificial Intelligence - An International Journal, vol. 33, pp. 89-103, 1987.
[22] M. Hammori, J. Herbst, and N. Kleiner, "Interactive Workflow Mining," presented at Business Process Management (BPM 2004) (vol. 3080 of LNCS), Potsdam, Germany, 2004.
[23] P. J. Barclay, T. Griffiths, J. McKirdy, N. W. Paton, R. Cooper, and J. B. Kennedy, "The Teallach tool: using models for flexible user interface design," presented at Proceedings of the third international conference on Computer-aided design of user interfaces, Louvain-la-Neuve, Belgium, 1999.
[24] H. Balzert, "From OOA to GUI -- The JANUS-System," presented at Proceedings of INTERACT'95, London, UK, 1995.
[25] A. R. Puerta and D. Maulsby, "Management of Interface Design Knowledge with MOBI-D," presented at In Proceedings of IUI'97, Orlando, FL, 1997.
[26] T. Elwert and E. Schlungbaum, "Modelling and Generation of Graphical User Interfaces in the TADEUS Approach," presented at Designing, Specification and Verification of Interactive Systems, 1995.
[27] P. Luo, P. Szekely, and R. Neches, "Management of interface design in HUMANOID," presented at Proceedings of InterCHI'93, 1993.
[28] E. Schlungbaum, "Model-based User Interface Software Tools - Current state of declarative models," Graphics, Visualization & Usability Center, Georgia Institute of Technology, Atlanta, GA GIT-GVU-96-30, 1996.
[29] P. Pinheiro da Silva, "User Interface Declarative Models and Development Environments -- A Survey," presented at Proceedings of DSV-IS2000, Limerick, Ireland, 2000.
[30] T. Griffiths, J. McKirdy, G. Forrester, N. Paton, J. Kennedy, P. Barclay, R. Cooper, C. Goble, and P. Gray, "Exploiting Model-Based Techniques for User Interfaces to Database," presented at Proceedings of Visual Database Systems (VDB) 4, Italy, 1998.
[31] F. Paternò, "One Model, Many Interfaces," presented at CADUI'02, Valenciennes, France, 2002.
[32] Q. Limbourg, "Towards Uniformed Task Models in a Model-Based Approach," 2001.