Accepted for the *Workshop on Practical UML-Based Rigorous Development Methods -- Countering or Integrating the eXtremists*, to be held in the scope of *<<UML>>2001- Modeling Languages, Concepts and Tools*. October 1-5, 2001, Toronto, Ontario, Canada.

Calculating with Concepts: a Technique for the Development of Business Process Support

Remco M. Dijkman[1,3] Luís Ferreira Pires[2] Stef M.M. Joosten[1,3]

[1] Open University of the Netherlands
P.O. Box 2960
6401 DL Heerlen
the Netherlands

[2] University of Twente
P.O. Box 217
7500 AE Enschede
the Netherlands
pires@cs.utwente.nl

[3] Ordina Finance Utopics
BU Anaxagoras
Hazenweg 88
7556 BM Hengelo
the Netherlands
{dijkman|joosten}@anaxagoras.com

Abstract: This paper introduces the Calculating with Concepts (CC) technique, which has been developed to improve the precision of UML class diagrams and allows the formal reasoning based on these diagrams. This paper aims at showing the industrial benefits of using such a formal and rigorous approach to reason about business processes and software applications in the early phases of the software development process. The paper discusses how the CC technique can be used in the specification of business processes and in the development of their supporting software applications or tools. This paper also illustrates the use of the technique with a realistic case study on tool integration.

1 Introduction

Business processes, like the handling of insurance claims or the closing of mortgages, have to be precisely defined, understood, and thoroughly analysed before software applications for supporting them can be developed. This implies that business processes have to be specified using techniques that are appealing to the business process architects, that enforce precision and clarity, and that allow analysis and manipulations. Furthermore, software applications to support business processes have to be specified in a similar form as these business processes, so that their suitability for supporting these processes can be evaluated or validated.

There are many graphical and formal modelling techniques in software engineering that can be used in the specification of distributed systems, in particular business processes and software applications. Graphical modelling techniques, such as, e.g., Entity Relationship (ER) [Ch76], have become popular for information analysis, data modelling and

the automated generation of data structures, because they are intuitively appealing and relatively easy to use. However, most graphical modelling techniques, such as, e.g., some diagrams of the Unified Modelling Language (UML) [WK99], lack a precise definition of their semantics [Ev98a, Ba96, KC00]. Consequently, different people may interpret graphical models drawn using these techniques in different ways. Different interpretations can lead to misunderstanding, unfruitful discussions or software implementations that do not comply with their requirements. This pleads for enhancing the precision of these techniques.

Formal modelling techniques, such as, e.g., Z [WD96], typically have precise syntax and semantics, defined in terms of mathematical models. This precision allows a rigorous approach to specification and reasoning. However, practitioners normally find these notations hard to use, mainly if their rather complex textual syntax is the only vehicle to produce specifications. This explains why formal notations are less used than they should be.

This paper introduces the Calculating with Concepts (CC) technique [JP00,Dij01], which has been developed to improve the precision of UML class diagrams. The goal of the CC technique is to make precise reasoning about functional specifications accessible to a large group of people. Therefore the technique has to be kept simple. This paper shows how the CC technique can be used to reason about functional specifications of business processes and software applications formally, i.e., with the aid of mathematical rigorousness. The paper discusses the applications of this technique in the modelling of business processes, and the development of software applications and the tailoring of tools to support them. The benefits of the CC technique are illustrated in this paper with a case study on tool integration.

The remainder of this paper is organised as follows: Section 2 introduces the CC technique, Section 3 discusses the applications of the CC technique that we have already identified in the area of business process architecture, Section 4 presents a case study on the use of the CC technique for tools integration, and Section 5 presents our conclusions and ideas for future work.

2 The Calculating with Concepts technique

This section defines the formal interpretation of UML class diagrams that is used in the CC technique, and introduces the cycle chasing technique, which allows one to identify and describe constraints on the universe of discourse described by the class diagram.

2.1 Conceptual modelling

We consider UML class diagrams as the common language for a group of stakeholders involved in a software development project. A UML class diagram describes a universe of discourse in terms of the different classes of objects that can be found there. Typical examples of classes are customer, mortgage file, savings account and letter.

A modeller constructing a UML class diagram relates classes by means of associations, such that each association stands for a sentence template in the universe of discourse. For example, by drawing a line between the classes Customer and Letter, and calling this association 'received', one can represent all instances of the sentence template 'Customer

<C> has received the letter marked <M>'. An example of instance of this template is 'Customer Brown has received the letter marked SJ01564', which relates a specific customer object (Brown) to a specific letter object (SJ01564). The modeller thus constructs a class diagram by drawing lines between classes, such that each line represents a different language template. Therefore, a class diagram describes a language with which sentences that are meaningful to a specific group of stakeholders can be represented.

Figure 1 shows a class diagram that represents the authorisation of employees to access customer files.

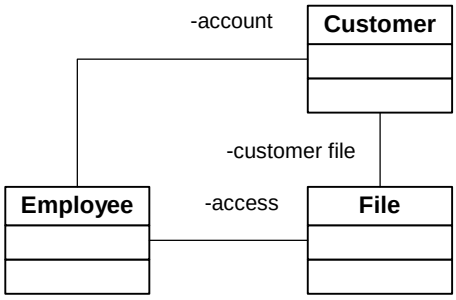

Fig. 1: Class diagram representing the access of an employee to customer files.

With the CC technique, we use UML class diagrams to generate models from a *conceptual perspective* [FS97], which means that we use class diagrams to define concepts and their relationships. Normally, one can specify attributes and operations in a UML class diagram, but since we concentrate on the conceptual perspective we do not support attributes and operations in the CC technique. Models from the conceptual perspective are called *conceptual models*. In later phases of the software development process, these models are normally used as requirements for the development of more detailed models that define the structure of the software application (software architecture).

2.2 Interpretation of a class diagram

The semantics of UML class diagrams is defined in the CC technique in terms of set theory and relational algebra. The user of the CC technique can choose either to translate class diagrams to the formal CC models and reason with these models, or to leave the class diagrams as they are, but adhere to the interpretation that is discussed in this section. The formal notation requires tool support since it is not appealing to human users, while the graphical form can be useful for communicating the models to stakeholders.

Each class and each (binary) association of a UML class diagram is interpreted as a set. The elements of each set represent the instances of the corresponding class or association. We postulate that there is a set of generic *objects* that represents all possible instances of all classes. This allows us to define the interpretation of classes and relationships formally, in terms of sub sets of this set. Therefore we define the interpretation of a UML class diagram *M* as the tuple

$M = < C_M, A_M, \text{restrictions}_M >$

where:

- $C_M \subseteq \wp$ (*objects)* represents the set of classes of model *M* ($\wp$ indicates a power set). Throughout this paper we use the variables *A, B* and *C* when we refer to an element of C_M. We use a, a_1, a_2 and b, b_1, b_2 and so forth to denote elements of *A* and *B*, respectively. Furthermore, we define that elements of C_M can only overlap if one is a subset of the other, i.e., $\forall A, B: A \cap B = \varnothing \vee A \subseteq B \vee B \subseteq A$;
- A_M represents the set of associations of model *M*. Throughout this paper we use the variables *R*, *S* and *T* when we refer to an element of A_M. We use r, r_1, r_2 and s, s_1, s_2 and so forth to denote elements of *R* and *S* respectively. An association *R* between classes *A* and *B* is represented by $R: A \rightarrow B$, which means that $(a, b) \in R \Rightarrow a \in A \wedge b \in B$. In our interpretation of class diagrams, associations have a direction, i.e., they are asymmetric. We indicate the direction of an association in the class diagrams by placing its name close to the target class;
- restrictions_M represents the set of restrictions of model *M*. Each restriction is represented as a predicate that holds for all populations of *M*.

Each $R: A \rightarrow B$ can be constrained by one or more of the following restrictions:

total (R)	$\equiv \forall a: \exists b: (a, b) \in R$
functional (R)	$\equiv \forall a, b_1, b_2: (a, b_1) \in R \wedge (a, b_2) \in R \Rightarrow b_1 = b_2$
surjective (R)	$\equiv \forall b: \exists a: (a, b) \in R$
injective (R)	$\equiv \forall a_1, a_2, b: (a_1, b) \in R \wedge (a_2, b) \in R \Rightarrow a_1 = a_2$

These restrictions are used to express the multiplicity constraints of a UML class diagram. Figure 2 shows the correspondence between multiplicity constraints in a UML class diagram and the corresponding restrictions.

Class diagrams often contain equivalence, inheritance and aggregation associations. These associations have properties that we can express as follows, for $R: A \rightarrow B$

equivalence	$A = B$
inheritance	$A \subseteq B$
aggregation	$A \subseteq \cup B$

Both equivalence and inheritance associations relate each element of *A* to the element of *B* to which it is equivalent, which can be represented as $(a, b) \in R \Rightarrow a = b$. Aggregation associations represent an element/set relation, which can be properly expressed using a mathematical part/whole relation. Therefore, we choose to represent each aggregate as a set of parts. Considering a simplified example of a car, it could be viewed as a set consisting of an engine and four wheels, represented as $car_1 = \{engine_1, wheel_1, wheel_2, wheel_3, wheel_4\}$. For an aggregate, the existence of its parts depends on the existence of the whole, which implies that these parts do not exist without the aggregate. This is expressed by the property defined above, since an aggregation relation relates each element of *A* to the elements of *B* of which it is a part, i.e., $(a, b) \in R \Rightarrow a \in b$.

When using the semantics described above, an easy to read formal specification can be derived directly from a UML class diagram. We currently have tool support to derive a formal specification from a UML class diagram both in Z, and in predicate logic.

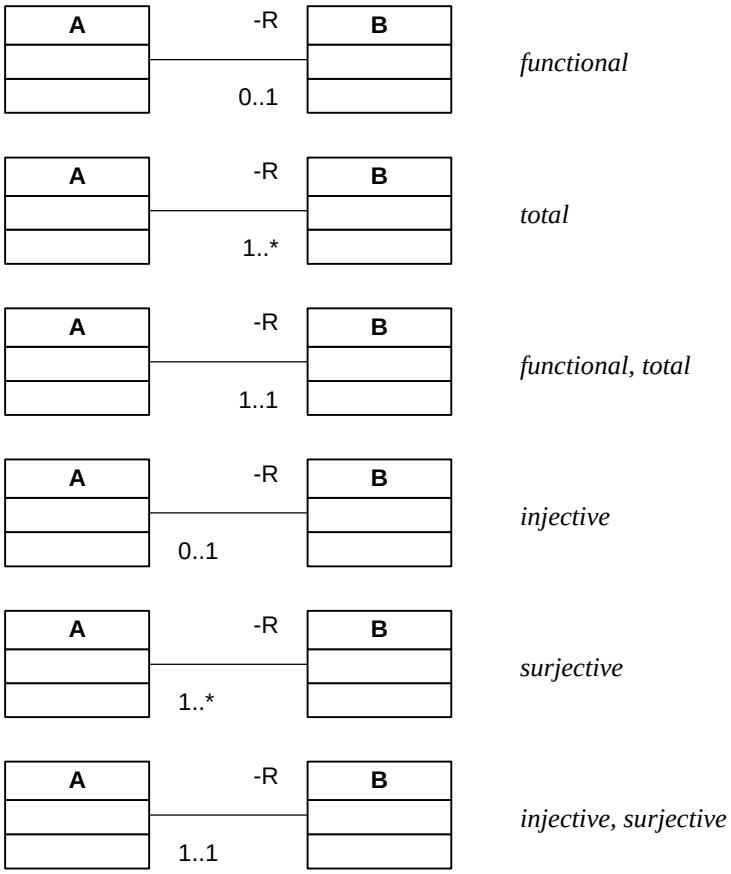

Fig. 2: Multiplicity constraints and their corresponding restrictions

2.3 Adding restrictions

In addition to objects and sentences, a universe of discourse may have some laws that constrain the number of valid instances of a sentence template. In Figure 1, for example, we could have the following law: ‘an employee has access to a file, provided that this employee is the account manager of the customer who owns the file’.

Cycle chasing is a technique that consists of following alternative paths determined by the associations between classes in a UML class diagram until a cycle is found, and verifying if the associations in the cycle can be constrained by a law. We discovered the law mentioned above by applying this technique to the class diagram in Figure 1. In this case, the cycle is formed by two paths: (1) 'access', and (2) 'account' followed by 'customer file'. We also suppose that the stakeholders have confirmed that such a law indeed constrains the model. Path 1 says which employees have access to which files, and path 2 defines a composed association that relates account managers to the files of their clients.

Laws are expressed in a CC-model by means of restrictions. The OCL [Ob97a] notation could have been used as an alternative to represent these restrictions.

Consider two relations R: $A \rightarrow B$, and S: $B \rightarrow$ C. In order to be able to represent laws that can be found by chasing cycles, we define two operators on relations:

inverse $\quad R^{\sim} = \{(b, a) \mid (a, b) \in R\}$

composition $\quad R ; S = \{ (a, c) \mid \exists b: (a, b) \in R \wedge (b, c) \in S \}$

The law we identified in the example in Figure 1 can be represented by

$access \subseteq account ; customer\ file^{\sim}$

Only for comparison, an OCL expression that describes the same law could be:

Employee
self.customer.file->includesAll(self.file)

3 Usage in process architecture

This section discusses the applications of the CC technique to business process architecture that have been identified and worked out in [Dij01]. These applications are concerned with improving design methodologies, integrating applications, and evaluating the functionality of software applications. Each one of these uses of the CC technique is briefly addressed below.

3.1 Improving methodologies

From our experience applying the CC technique to case studies in [Dij01] we learned that the following principles, which can be applied in design methodologies, play an important role in software development projects:

- inter-relation dependencies, such as the dependencies that can be identified using cycle chasing (see Section 2.3), may contain important design information. The capability of representing inter-relation dependencies explicitly improves the expressive power of the modelling technique;
- models can be generated at different abstraction levels, depending on the stakeholders (users, architects, implementers) or the purpose of the model. One should be capable of relating these different models, amongst other reasons, to produce a common language for communication between stakeholders. In order to achieve this we can define mathematical relations between the classes of the different models, and restrictions involving the associations of the different models, so that the correspondence between the models is formally represented;
- the explicit distinction between a definition part and an execution part in a business process model is normally essential to guarantee the common understanding of the concepts in a model.

Figure 3 illustrates the definition part of a model (Activity definition and Connection concepts) and the execution model (Activity and Trigger). In this example a person is responsible for activities of some type, and the corresponding activity instances once they are created. Activity types are connected to each other in the definition part of the model, while an activity instance may trigger another activity in the execution part of the model.

The CC technique in this example plays an important role since cycle chasing can be used to keep the type consistency between the definitions and the instances. We can, for example, express the restriction that a trigger can only cause an activity, if this trigger is an instance of a connection to the activity's type:

$causes \subseteq type \; ; \; to \; ; \; type^{\sim}$

Candidates for inter-relation dependencies can be identified using cycle chasing, as illustrated in Section 2.3. The use of the CC technique for relating models at different abstraction levels will be tackled in a forthcoming paper.

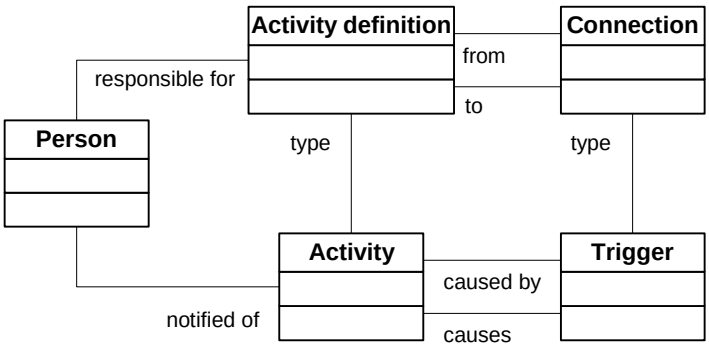

Fig. 3: Definition and execution parts of a model

3.2 Applications integration

In most cases, business processes are not really supported by a single software application, but by a set of applications. These applications are normally developed separately, possibly by different development teams or companies, and are integrated later to form the automated support that is required. The major challenge of this integration task is to assure that the resulting support is consistent and complies with the requirements dictated by the business process. The CC technique allows one to formally define relationships between concepts of the applications to be integrated, so that consistency rules can be defined and allowing the consistency of the resulting support to be assessed. Section 4 describes in more detail a case study on tool integration using the CC technique.

3.3 Functionality evaluation

Sometimes we have to evaluate whether some legacy piece of software can be replaced by a more advanced tool or application. Users are normally reluctant when doing that because they feel comfortable with the legacy software that they know well and do not want to learn something new, or because they are afraid that they will lose functionality. In [Dij01] we show that the CC technique can help users getting confidence that the replacement of legacy software will not imply loss of functionality. This can be achieved by making models of both the legacy software and its intended replacement, and defining formal mappings from concepts and associations of the model of the legacy software onto corresponding concepts and associations of the model of the intended replacement. While defining the correspondence, which gives already design information for imple-

menters, one can evaluate the suitability of the replacement, generating documentation that can be used to convince the user. This usage of the CC technique will also be reported in more detail in a forthcoming paper.

4 Case study on tools integration

The software support to business processes can consist of a set of applications or tools, which are developed separately and integrated afterwards. In order to integrate these applications or tools in a proper way according to the requirements of the business processes we have to evaluate these applications or tools at a conceptual level. This evaluation should reveal the alternatives for integration and whether the integration is possible at all. In the sequel we show that CC-technique can help performing this evaluation.

4.1 Formal definitions

In order determine how two subsystems, represented by the CC models *M* and *N*, can be integrated, we define a third CC model *G* (for glue), which contains the associations between classes in *M* and classes in *N*. *G* also contains the restrictions that range over these associations.

After having determined *G*, we assemble the CC models using the $\cup$ operator, which is defined for two CC models *M* and *N* as:

$$M \cup N = < C_M \cup C_N, A_M \cup A_N, \text{restrictions}_M \cup \text{restrictions}_N >$$

The technique used to assemble CC models resembles the principles of database view integration [BLN86], and the principles of knowledge-base integration [BK99]. After the integration has been performed, the resulting model should be evaluated, to check if it is realistic.

4.2 Protos and Staffware

A case study reported in [Be00] discusses the integration of the process modelling tool Protos [Pa98] with the workflow tool Staffware [St96], allowing business processes modelled using Protos to be supported by Staffware. In this case study the CC models of both tools were assembled, proving that the most straightforward translation from a Protos model to a Staffware model leads to useless workflow designs, thereby preventing a costly mistake to be made. The mapping between these tools that leads to useful workflow designs was much more sophisticated and could be better represented using a formal model such as the CC technique.

4.3 Resulting integration

Figure 4 illustrates the use of the CC technique in this case study in a strongly simplified form. Staffware handles the co-ordination of work using the classes procedure, step and group. A step is a unit of work that can be performed by a single person in a singular unit of time like, for example, 'prepare interview with customer'. A procedure is a collection of work related to a specific goal, like 'loan application', which is the collection of work related to the goal of selling a loan. Typically the work involved in a workflow procedure is described as a series of steps. The procedure 'loan application' contains, for ex-

ample, the steps 'prepare interview with client', and 'carry out interview with client'. A group is a set of people that are similar with respect to the tasks they perform, like 'client advisors'.

Protos models business processes using the classes procedure, activity and role. The definitions of procedure and role are similar to the definition of procedure and group in the workflow pattern. An activity is the smallest unit of work that makes sense to a person, like 'check that loan amount does not exceed limit'. These classes and their associations have been captured in the solution part of the business process pattern.

From the definitions of the classes in the two original models, we conclude that the workflow class procedure has an equivalence association to the business process class procedure. The role and group classes also hold an equivalence association. The step class has an aggregation association to the activity class, because activities that can be carried out by a single person in a singular unit of time, can be grouped into steps. The activities 'check that loan amount does not exceed limit' and 'write proposal' can, for example, be grouped into the step 'prepare interview with client'. The assembly of the two patterns, by the associations, results in new cycles from which we derived the following laws:

- two activities are in the same step, provided that they are performed by the same role or group; and
- a step follows another step, provided that the first step contains an activity that follows an activity in the second step.

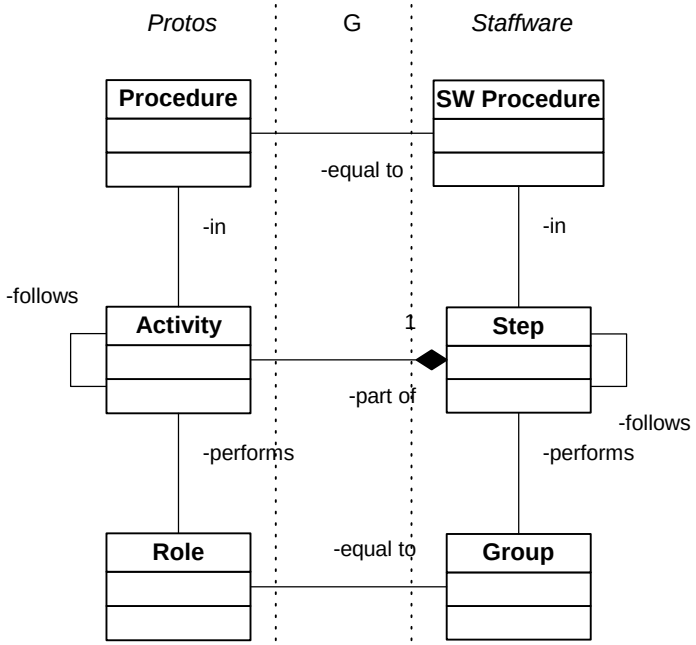

Fig. 4: Combination of Staffware and Protos

These laws are represented by the following restrictions:

functional(*part of*$^{\sim}$; *performs*$^{\sim}$)
follows $\subseteq$ *part of*$^{\sim}$; *follows* ; *part of*

The glue model *G* thus is composed of the following elements:

C_G = { *Procedure*, *SW Procedure*, *Activity*, *Step*, *Role*, *Group* }
A_G = { *equals*: *Procedure* $\rightarrow$ *SW Procedure*, *part of*: *Activity* $\rightarrow$ *Step*,
equals: *Role* $\rightarrow$ *Group* }
$restrictions_G$ = { *functional*(*part of*$^{\sim}$; *performs*$^{\sim}$),
follows $\subseteq$ *part of*$^{\sim}$; *follows* ; *part of* }

We have performed the integration by applying the $\cup$ operator to the formalised Staffware and Protos models, and *G*. Figure 4 shows a UML class diagram of the assembled workflow and business process CC model.

We have developed tool support for formalising class diagrams into CC models, and for assembling them. This tool delivers formal specifications in Z. Figure 5 shows the Z specification of the integration of Staffware (represented as *Workflow*) and Protos (represented as *Proceduremodeling*).

$Workflow \cup Proceduremodeling \cup Glue$

$Procedure : \mathbb{P}\ objects$
$Activity : \mathbb{P}\ objects$
$Role : \mathbb{P}\ objects$
$SW\ Procedure : \mathbb{P}\ objects$
$Step : \mathbb{P}\ objects$
$Group : \mathbb{P}\ objects$
$in : SW\ Procedure \leftrightarrow Step$
$in : Procedure \leftrightarrow Activity$
$follows : Activity \leftrightarrow Activity$
$follows : Step \leftrightarrow Step$
$performs : Role \leftrightarrow Activity$
$performs : Group \leftrightarrow Step$
$part\ of : Step \leftrightarrow Activity$
$equal\ to : Procedure \leftrightarrow SW\ Procedure$
$equal\ to : Role \leftrightarrow Group$

$follows \fatsemi in^{\sim} \subseteq in^{\sim}$
$follows \subseteq part\ of^{\sim} \fatsemi follows \fatsemi part\ of$
$functional(part\ of^{\sim} \fatsemi performs^{\sim})$

Fig. 5: Z specification of the integration of Staffware and Protos

This example shows that the CC technique helps defining the rules for integrating two applications (tools) and provides the machinery to reason about the resulting integration from the perspective of the requirements of the business processes.

5 Conclusions

In this paper we introduce the Calculating with Concepts (CC) technique. The CC technique provides a precise basis for UML class diagrams, aimed at simplifying and supporting reasoning about designs in an industrial environment. To do so, the CC technique specifies a formal semantics for UML class diagrams, and a technique called cycle chasing, for identifying additional restrictions to the resulting models.

The paper identifies a number of applications of the CC technique in which this technique is used as a reasoning tool. The paper shows that the CC technique can be used to relate the languages spoken by different stakeholders in an organisation, thereby helping them to communicate. We also show that the technique can be used to assess if a tool can be used to replace a legacy system, assessing if, and how, functionality is translated from the legacy system to the replacement tool. The CC technique is used to specify the relation between tools that together support a business process, and to assess if this relation results in a consistent system. The paper also shows how the relation between the tools can be specified in Z, thereby providing a formal specification of the customisation necessary to integrate the tools.

The CC technique has been applied in a number of practical situations. This paper shows an example from practice, where the application of the CC technique proved that the integration of two tools was wrongly defined, preventing that a costly mistake was made. We also applied the CC technique in two other situations, in which two consultants, with relatively little experience, where asked to review a functional design. The CC technique helped them improving the designs significantly, proving itself useful.

The work described in this paper is strongly related to the work of the precise UML (pUML) group [Ev98a, Ev98b, Ev98c, Ev99, BF98]. This group, however, aims mainly at describing a formal semantics for UML, while we aim at making formal reasoning with UML models accessible to a large group of people. Therefore our semantics must be easy to learn, and hence it should remain simple. Many references can be found in the literature in which the generation of formal models from informal conceptual models from the software development point of view is addressed [KC00, Ba96, MP95, DLC00]. From these, only [DLC00] addresses the integration of formal restrictions in conceptual models. The latter reference, however, does not go as far as using the resulting formal specifications for reasoning. Apparently not much has been published on the integration of tools at a conceptual level.

Further research related to the CC technique will be mainly done in three areas: (1) the formalisation of the dynamic aspects of systems, since the work so far only addresses the static aspects; (2) the development of tool support to perform evaluations on assembled models, and (3) more practical applications of the technique, by defining more operators on CC models and identifying design questions that can be answered with the help of the CC technique. In future, we also intend to apply the CC technique to the field of design patterns. We will use the CC technique to formalise design patterns and integrate them systematically into designs of complete systems.

Bibliography

[Ba96] Bates, B.; Bruel, J.; France, R.; Larrondo-Petrie, M.: Formalizing Fusion Object Oriented Analysis Models. In: Proceedings of the First IFIP International Workshop on Formal Methods for Open Object-based Distributed Systems. (1996)

[Be00] Beek, J. van: Generation Workflow - How Staffware Workflow Models can be Generated from Protos Business Models. Master's thesis, University of Twente (2000)

[BF98] Bruel, J.-M.; France, R.: Transforming UML Models to Formal Specifications. In: Proceedings of the OOPSLA'98 Workshop on Formalizing UML: Why? How? (1998)

[BLN86] Batini, C.; Lenzerini, M.; Navathe, S.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys **18**, No. 4, 323-364 (1986)

[BK99] Borgida, A.; Kuesters, R.: What's not in a name? - Initial Explorations of a Structural Approach to Integrating Large Concept Knowledge-Bases. Technical report DCS-TR-391, RWTH Aachen (1999)

[Ch76] Chen, P.: The Entity Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems **1**, No. 1, 9-36 (1976)

[Dij01] Dijkman, R.: Calculating with Concepts - A Formal Conceptual Modelling Technique Applied in the Field of Process Architecture. Master's thesis, University of Twente (2001). Avaliable at: http://arch.cs.utwente.nl/ → Assignments

[DLC00] Dupuy, S.; Ledru, Y.; Chabre-Peccoud, M.: An Overview of RoZ - A Tool for Integrating UML and Z Specifications. In (Wangler, B.; Bergman, L.): CaiSE 2000. Lecture Notes in Computer Science 1789, 417-430, Springer Verlag (2000)

[Ev98a] Evans, A.: Reasoning with UML class diagrams. In: WIFT '98 Proceedings, 1998. IEEE (1998)

[Ev98b] Evans, A.; France, R.; Lano, K.; Rumpe, B.: The UML as a Formal Modelling Notation. In (Bézivin, J.; Muller, P.): The Unified Modelling Language - UML '98: Beyond the Notation: First International Workshop. Lecture Notes in Computer Science 1618, Springer-Verlag (1998)

[Ev98c] Evans, A.; Bruel, J.-M.; France, R.; Lano, K.; Rumpe, B.: Making UML Precise. In (Andrade, L.; Moreira, A.; Deshpande, A.; Kent, S.): Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How? (1998)

[Ev99] Evans, A.; Kent, S.: Core meta Modelling Semantics of UML - The pUML Approach. In (France, R.; Rumpe, B.): The Unified Modelling Language - UML '99: Beyond the Standard: Second International Conference. Lecture Notes in Computer Science 1723, Springer-Verlag (1999)

[FS97] Fowler, M.; Scott, K.: UML Distilled - Applying the standard object modeling language. Addison-Wesley (1997)

[JP00] Joosten, S.; Purao, S.: A Rigorous Approach for Mapping Workflows to Object-Oriented IS Models. Submitted to Journal of Database Management (2000)

[KC00] Kim, S.-K.; Carrington D.: A Formal Mapping between UML Models and Object-Z specifications. Technical Report 00-03, Software Verification Research Center University of Queensland (2000)

[MP95] Mander, K.; Polack, F.: Rigorous specification using structured systems analysis and Z. Information Software and Technology **37**, No. 5-6, 285-291 (1995)

[Ob97b] Object Management Group, UML Notation Guide. Object Management Group (1997)

[Pa98] Pallas Athena: Protos 3.0 Handleiding. Manual, Pallas Athena, Plasmolen (1998) (in Dutch)

[St96] Staffware: Staffware Workflow - Client User Manual. Manual, Staffware, Berkshire (1996)

[WK99] Warmer, J.B; Kleppe A.G.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (1999)

[WD96] Woodcock, J.; Davies, J.: Using Z - Specification, Refinement, and Proof. Prentice Hall, London (1996)