Gaining a Uniform View of Different Integration Aspects in a Prototyping Environment*

J. Kusch‡ P. Hartel† T. Hartmann† G. Saake‡

Abstract: Prototyping based on formal specification should satisfy the requirements of engineers and users equally. The support of both groups requires a formalization of all activities as contract base and an increase of experience by experimental strategies. Contrary to these requirements several problems rest with existing software development environments, which result in complex systems becoming easily unmanageable. Our approach to develop a prototyping environment integrates former results of our project "Implementation of Information Systems". We propose the object-oriented specification language TROLL as core of the integrated software development environment *TBench*. TROLL is used for conceptual modeling of structural and behavioral aspects of the universe of discourse by offering a formal framework. Moreover the *TBench* utilizes TROLL to realize relationships between the various graphical representations and the different stages of development. Especially TROLL specification documents cover the aspects of integration on data, control and process.

1 Introduction and Motivation

Most failures in software projects can be traced back to a lack of precise description of the conceptual model. During development of complex systems, the claims of users as well as engineers meet in the following two extremes [BKKZ92]: (1) *Engineers* need a precise and abstract description of the conceptual model [Gri82]. As base of a contract, the possibility of verification [Con94] of specified assertions must be supported. (2) *Users* need to validate the conceptual model by animation [HJS93] to decide whether the result corresponds to the given instructions. Prototyping [BP92] on base of object-oriented specification unites these requirements. Therefore *tool support* is inevitable.

Our approach of a prototyping environment is based on former results of our project "Implementation of Information Systems": Objectives of the project have been the abstract description and validation of a specified universe of discourse.

‡Institut für Technische Informationssysteme, Otto-von-Guericke-Universität Magdeburg, D–39106 Magdeburg, Germany, E-Mail: {kusch|saake}@iti.cs.uni-magdeburg.de

†Institut für Programmiersprachen und Informationssysteme, Technische Universität Braunschweig, D–38092 Braunschweig, Germany, E-Mail: {hartel|hartmann}@idb.cs.tu-bs.de

*This work was partially supported by the CEC under ESPRIT-III BRA WG 6071 IS-CORE and ESPRIT BRA WG 8319 ModelAge. The work of T. Hartmann and J. Kusch is supported by DFG under Sa 465/1-3. The work of P. Hartel is supported by OBLOG Software SA, Lisboa, Portugal.

Thus tool support for specification and animation is mandatory. Results have been the object-oriented specification language TROLL [HSJ+94] which is based on a formal framework [ESS92, FSMS91]. For conceptual modeling, information systems are modeled as object societies [JSHS95, SJH93]. For specification support, a graphical notation OMTROLL is proposed in [JWH+94]. OMTROLL is a marriage of the formal specification language TROLL and the popular method OMT [RBP+91]. A combination of both allows to view the different aspects of the system in an integrated manner [Bri93]. The resulting prototyping environment *TBench* integrates these results by using TROLL multi-functional as core of the ISDE: (1) The universe of discourse is specified conceptually. (2) Different representation aspects (graphical and textual) are integrated as conceptual entirety. (3) The formal framework enables verification of conceptual properties. (4) Different development phases are combined by prototyping for validating the specified object society. (5) Aspects of control and process integration within an ISDE are maintained as TROLL specification documents. This keeps making the *TBench* a process driven and easy to manage ISDE which could be configured via its specification tools.

2 TROLL and OMTROLL

Modeling of object societies is divided into graphical specification of global information (OMTROLL) and textual refinement of details (TROLL). Both techniques are required for the modeling of complex information systems.

TROLL – **Core of the ISDE** ***TBench*** Conceptual modeling of information systems requires the description of the application domain, the so-called *Universe of Discourse* (UoD), on a high abstraction level. Looking at an information system (and its environment), a collection of interacting *objects* seems to be a very natural way for conceptualizing information structures and processes. Objects have a local state, show a specific behavior, communicate with other objects and may be itself composed from smaller objects.

A language supporting the conceptual modeling phase has to provide a variety of abstraction principles as there are for example classes, specialization/generalization hierarchies (roles), composite objects, relationships and views. These principles of object descriptions are represented in TROLL [JSHS95, HJS93, SJH93] as *encapsulated specification items* (ESpec), which are elements of an object society. TROLL ESpecs are structured as a two level hierarchy of *concepts* and *features*. This way we achieve in having a clustered description of relevant properties of objects. Details on the specification language TROLL are illustrated in [HSJ+94].

OMTROLL – **Integration of different Views** Quite often in the analysis of systems, different aspects are expressed using various methods and formalisms, e.g. ER models for the static aspects, Petri-Nets for systems dynamics and data flow diagrams to express computations. Object-oriented methods overcome this heterogeneity by providing the notion of object which allows the specification of state and dynamics in one concept. But there still remain problems, like some object-oriented methods are not suited for all aspects of systems and most popular object-oriented methods are rather informal and vague in their definition. We do want to overcome these limitations by the integration of formal and informal methods to build a homogeneous specification framework.

Object-oriented analysis methods have a background in system development practice. Formal languages are based on academic research mainly and are only now beginning to be applied to practical software development projects. This suggests that a marriage between the two approaches to modeling and development could be fruitful. There are a number of advantages to be gained by such a marriage: (1) It could help in discovering and eliminating ambiguity and vagueness in the informal representation techniques. (2) Combining a practical analysis method with a formal specification language helps in making the formal specification language fit for use. (3) Formal specification languages usually come with logical inference systems, and combining this with a practical analysis method makes powerful techniques available for analysis. We made the attempt to integrate the Object Modeling Technique [RBP+91] with the TROLL language [HSJ+94] to a method which we call OMTROLL [JWH+94] (Figure 1). The interested reader is referred to [JWH+94] for OMTROLL and [RBP+91] for OMT.

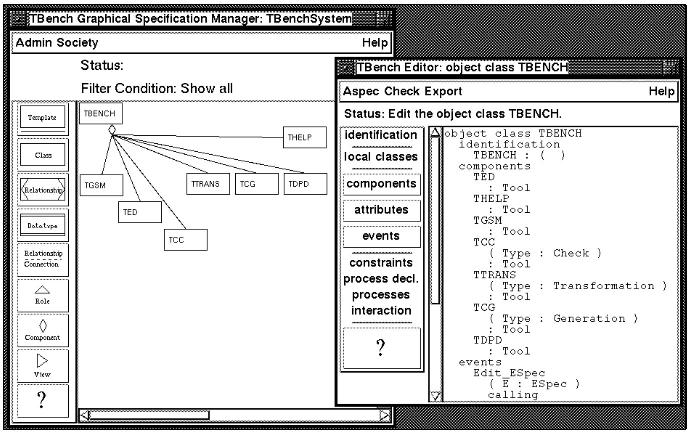

Fig. 1

3 Integration Aspects

The consideration of integration aspects for software development environments [SB93] leads to consistent inter-connections between the components of a software development process which should be adequately embedded within a manageable ISDE. In general development environments consists of a set of tools and an integration frame [Cag90] which could be partitioned into the components database system, tools, user interface and development process. The interfaces between this components are defined by the ECMA reference model. Furthermore the granularity of integration is measured by levels from container-level up to method-level [BM92], which serves to assess integration aspects of ISDE. We pay attention to integration aspects [TN92] on *data*, *control* and *process*. Due to the fact, that TROLL object specifications do not regard the modeling of graphical user interfaces, we do not emphasize on *presentation* integration.

3.1 Data Integration

The *TBench* provides *data integration on syntactical level*, which is obtained by abstract public known data structures.

***TBench* Concepts of Data Integration** The centralized *Repository*, which is part of the TIM, persistently maintains all relevant information referenced to a software development project. That are mainly *specification data* (TROLL specification documents) but also *configuration and administration data* (ie. public services, location information for data and tools in a distributed environment, access rights, lockings) and *animation data* (ie. long transaction, runtime information). To decrease complexity and enhance learnability all above mentioned information are managed as TROLL specification documents. Especially control and process integration information are introduced in detail in section 3.2 and 3.3. This uniform document format in shape of the *TBench* object society `TBenchSystem` enables configuration and controlling of the *TBench* via its specification tools.

Therefore the repository interface has to be based on the taxonomy of specification documents which is a hierarchical structure of object society, ESpecs, concepts and features. An information system is represented by an *object society*. Each object society is composed of a set of ESpecs (ie. object class, role, view, relationship, ...). Each *ESpec* again consists of a set of concepts (ie. attributes, events, components, interactions, ...) which structure an object specification. Each *concept* again consists of a set of features (ie. name of a concept, data type of a concept, derivation rule of a concept). Thus an ESpec could be represented through *three dimensions*: (1) Each ESpec consists of different concepts (ie. attributes and events). (2) An ESpec may contain several instances of the same concept (ie. an object has several attributes). (3) Each instance of a concept is composed of a variety of different features (ie. an attribute has a name, a data type and may be a restriction rule).

TROLL **Data Integration Structure** In opposite to a flat file, the usage of a database system offers several well known advantages. To simplify the implementation of a set of tools for the *TBench* our approach proposes (with respect to the data access profile of the tools) an abstract *data model independent* interface to the *TBench* Integration Manager. Each ESpec of an object society could be represented as an *heterogeneous 3-dimensional array*. Heterogeneity is caused by different data types for different TROLL features, which have to be maintained economically. The disadvantages of that primitive and uncomfortable data structure are eliminated by various enhancements: (1) Inherent persistency. (2) Transparent distribution on object society level. (3) Connection of underlying heterogeneous database systems. (4) Locking on ESpec level for multi-user and multi-project running. (5) Feature-Cache (Server initiated approach with FIFO strategy). (6) Garbage collection. (7) Centralized access optimization. (No data flow with check in/out in the conventional way. Only demanded data is transferred). (8) Access control and restriction. (9) Access logging for posterior project documentation and analysis. This data structure only stores feature information (as abstract TROLL syntax). TROLL keywords are generated afterwards by textual representation tools. As this enhanced array maintains all persistent data of the *TBench* the tools require no additional internal data structures.

Advantages of data integration on syntactical level are easy to implement "*memory less tools*". The integration frame of the TIM hides a couple of complex mechanisms for the tools. Furthermore the envisaged data interface enables parallel modification access on different fragments of the same ESpec. This is the base for parallel execution of prototyping services (section 3.3).

3.2 Control Integration

The *TBench* provides *control integration on semantic level*, which is obtained by public knowledge about offered services and its properties.

***TBench* Concepts of Control Integration** Each tool of the *TBench* is partitioned into a set of public services and non-public functionality. Tools are coevally clients and servers because any tool offers and claims public services. Services are administrated tool- and location-independent by the *TBench* Integration Manager (TIM) which acts in a transparent manner like a *request redirector*. That is the demands of a client are first of all directed to the local TIM. If the claimed service or data is not local, the request is forwarded to a TIM which could fulfill the query. The locally serving TIM knows about the tool which offers the claimed services. This is realized by a mapping from each public service (eg. Society_Display, Print_ESpec) to the serving tool and its location. Thus, each functionality only has to exist once in a network of tools. Functionality redundancy is avoided.

To give an example, a consistency checker tool (TCC) encounters an error in a TROLL specification document. The processed specification document is "buried" as an abstract data structure in the *TBench* repository. To display the point of inconsistency, the erroneous fragment has to be presented to the user in a TROLL like shape. Only the *TBench* Editor (TED) which achieves syntax sensitive specification and modification of ESpecs, knows (and has to know as the only one) about the textual representation of TROLL specification documents. Thus the consistency checker informs the TIM to print a fragment of an ESpec. The TIM redirects this demand to the TED, which finally performs the service.

TROLL **Control Integration Specification** The description, which covers all these required information, is specified in TROLL. Thus, by editing this specification document via the *TBench* specification tools, the ISDE could be extended and modified. Tools could be simply exchanged. The society `TBenchSystem` enumerates appropriate classes which contain all required information about configuration, administration (eg. classes *USERS*, *SCHEDULES*, *LOCKINGS*, *ACCESSRIGHTS*) and animation (eg. classes *ANIMATIONS*). Additionally information about each tool (eg. *TGSM*, *TED*, *TCC*, *TTRANS*, *TCG*, *TDPD*) are structured as object class specifications and modeled in an intuitive way as components of the class *TBENCH*. Especially the class *TBENCH* contains available public services and additionally the prototyping process which is describes in detail in section 3.3. Thus the information itself is depicted on the *object specification document level*, not on the instance level!

Via the data interface of the *TBench* repository (section 3.1) each tool could query the available public services which are specified as *events* of the class TBENCH. By means of the parameters of events, a tool could decide for what kind of specification item (eg. a whole society or an ESpec) the offered service is defined. The `calling` section of each event describes the particular tool and its functionality. The location information is specified within the class of the tool.

```
society TBENCHSYSTEM
  TBENCH, ACCESSRIGHTS,
  USERS, SCHEDULES, LOCKINGS,
  ANIMATIONS, ACTIVITIES,
  TGSM, TED, TCC, TTRANS, TCG,
  TDPD, EXECUTOR, TANIMATOR
end society TBENCHSYSTEM.

object TBENCH
  ...
  components
    TGSM: Tool
    TED: Tool
    TCC(Type: Check): Tool
    TTRANS(Type: Transformation): Tool
    TCG(Type: Generation): Tool
    TDPD: Tool
    ...
  events
    Display_Society(S: Society)
      calling TGSM.Display(S).
    Edit_ESpec(E: ESpec)
      calling TED.Edit(E)
    Check_References(S: Society)
      calling TCC('References').Check(S).
    Check_Reachability(S: Society)
      calling TCC('Reachability')
                              .Check(S).
    Transform_Temporals(S: Society)
      calling TTRANS('Temporals')
                              .Transform(S).
    Transform_Composites(S: Society)
      calling TTRANS('Composites')
                              .Transform(S).
    Generate_CPP(S: Society)
      calling TCG('CPP').Generate(S).
    Generate_Plg(S: Society)
      calling TCG('Prolog').Generate(S).
    Execute_Society(S: Society)
      calling EXECUTOR.start(S).
    ...
end object class TBENCH.
```

Advantages of control integration on semantic level are the avoidance of redundant implementation of functionality and dynamic extensibility of the *TBench*. Furthermore centralized guarding and logging of service-demands enables restriction of access to services and documentation of the development process for a mandatory succeeding analysis phase. The system information specified as TROLL object societies permits the configuration via specification tools.

3.3 Process Integration

The *TBench* provides *process integration on semantic level*, which is obtained by centralized logging of all activities of the software development process for public inquiry.

TBench **Concepts of Process Integration** At a higher level than data, control or presentation integration, process integration makes the implicit chain of development tasks explicit. This is the base for a process driven development environment [MS92]. A software process model (SPM) is a formal way to organize and describe how life-cycle models, development methods, software artifacts, CASE tools, and developers fit together – it is a collection of objects representing activities, artifacts, tools and developers. Our implementation strategy is to specify the SPM of the prototyping processes for the *TBench* by TROLL. This enables to control and modify the development by the available specification tools. Furthermore the developers only have to deal with TROLL, which enhances the learnability of the ISDE. By this the embedding of the software development process into the development environment is performed in an adequate way using the same integration paradigms as they are used to cover other aspects of integration.

TROLL offers a process sub-language to specify allowed life cycles over objects within an object society. This sub-language is a useful vehicle to model the development process as well. The TDPD interprets and executes the modeled SPM (TROLL process script [MBW80]), according to its activity hierarchy [WR92]. It manages the order of subtasks and the constraints to be satisfied before they can be performed. It plays the role of an automated driver in that it initiates an SPM, enacts its subtask partially in parallel, accepts developers inputs, updates and propagates the status of these subtasks and triggers other subtasks.

TBench **Prototyping Process** The prototyping process of the *TBench* pictures the connection from the graphical modeling of conceptual aspects of the universe of discourse to independently executable prototypes of applications. Input from the developer is only needed in the phase of specification. The further phases are performed automatically by the TDPD. Each of them requires the effect of its previous phase to be performed:

Consistency check: A set of consistency checkers (TCC – *TBench* Consistency Checker) control different aspects of consistency of one society. Aspects, which have to be considered are eg. existence of references between different classes, unobtainable specification fragments. Especially the support of the *TBench* repository enables parallelism for the performance of this phase.

Transformation: A set of transformators (TTRANS – *TBench* Transformator) map high level TROLL concepts into an operational language TROLL-kernel. TROLL-kernel is a subset of TROLL. It only contains abstractions that are coherent to concepts of common programming languages, like C++. Concepts of TROLL which have to be transformed are ie. temporal logic, object communication, properties of complex objects. Again this phase could be performed partially in parallel, if the transforming aspects are independent from each other.

Generation: Different generation tools (TCG – *TBench* code generator) map TROLL-kernel specifications to code of an existing object-oriented programming language (ie. C++, Sather). Nonfunctional requirements are supported by the use of additional tools (eg. E/Exodus). The result is a prototype of an independent application.

Execution: The generated prototypes enable validation [HJS93] of the specified conceptual model.

In coherence to the above mentioned phases the TROLL specification describes the prototyping process in a formal way.

TROLL **Process Integration Specification** The process declaration sublanguage of TROLL supports sequences of events and processes (denoted by '− >'), a choice operator (denoted by '|'), parallel execution (denoted by '‖') and preconditions (specified before an event in braces). Furthermore a process declaration could be clustered by subprocesses and events which are offered as public services by the *TBench*.

```
object class TBENCH
 ...
 attributes
  status: SPM_STATUS.
  ...
 events
  set_failure changing status=failure.
  reset_failure changing status=ok.
  ...
 process
  specification_cycle
    initiative interleaving free.
  prototyping_cycle
    initiative interleaving free.
  check_cycle
    interleaving excluding
                  specification_cycle.
  transformation_cycle
    interleaving excluding
                  specification_cycle.
  generation_cycle
    interleaving excluding
                  transformation_cycle.
  execution_cycle
    interleaving excluding
                  generation_cycle.
  ...
 process declaration
  specification_cycle(S: Society, D: Destin)=
    Display_Society(S) − >
    (specification_cycle(S) |
    prototyping_cycle(S, D) |
    exit).
  prototyping_cycle(S: Society, D: Destin)=
    reset_failure − >
    check_cycle(S) − >
    transformation_cycle(S) − >
    generation_cycle(S, D) − >
    execution_cycle(S).
  check_cycle(S: Society)=
    (Check_References(S) ||
    ... ||
    Check_Reachability(S)) − >
    ({status=failure} specification_cycle(S)).
  transformation_cycle(S: Society)=
    (Transform_Temporals(S) ||
    ... ||
    Transform_Composites(S)) − >
    ({status=failure} specification_cycle(S)).
  generation_cycle(S: Society, D: Destin)=
    ({D='C++'} Generate_CPP(S) |
    {D='Prolog'}) Generate_Plg(S)) − >
    ({status=failure} specification_cycle(S)).
  execution_cycle(S: Society)=
    Execute_Society(S).
 ...
end object class TBENCH.
```

The `process` section then specifies the interleaving in sense of database transaction theory (denoted by `interleaving`). Only processes which are not dependent from each other could interleave, thus where input and output data have no connection. Interleaving restrictions avoid update inconsistencies between different phases of the prototyping process. The initiation of the processes is driven by calling or by the user. E.g. the *edit_cycle* and the *prototyping_cycle* are initiated by the user itself (denoted by `initiative`). Other cycles are initiated by calling of other processes. The status attribute *SPM_STATUS* only denotes a rough status of the prototyping process. The `process declaration` section describes the particular phases of the prototyping process (e.g. *specification_cycle*, *check_cycle*). They are combined out of preconditions, sequence constructors and public services.

The advantages of process integration on semantic level are the possibility that tools can obtain "knowledge" about the actual phase of the development process. Public information of performed activities in conjunction with the prototyping process description guarantee to avoid redundant task execution within a process driven environment. Furthermore this is the adequate base to guide, guard, analyze and document a prototyping process.

4 The first Prototype

Our goal was to develop a prototyping environment which guides the straight way from graphical specification to independent executable prototypes. We had to consider *sound integration*, *extensibility/openness* and *portability* that help to manage the inherent complexity of the *TBench* implementation.

Due to the experiences of the OBLOG Workbench [Esp93], the first prototype of the development environment is based on a relational database system. Reasons for it have been portability through standardization, availability, available knowledge and reliability. The used platform is SunOS and Ingres. A port to *Solaris*, *Linux*, *Oracle* and *Yard* is under construction. Without changing any tool, the architecture of the *TBench* enables porting to an object-oriented database, which eventually will follow.

The tools and mechanisms which are used for the current implementation are C/C++ as programming language, RDBMS Ingres for the repository, Tk/Tcl for the graphical user interfaces, RPC-GEN for the inter-tool communication, Flex/Bison (GNU derivation of Lex/Yacc) for syntax checking and E/Exodus for the generated runtime modules. Figure 1 depicts a snapshot of the graphical development environment of the *TBench*.

5 Conclusions and Outlook

The *TBench* is a *partial result* of the DFG project "Implementation of Information Systems". In opposite to a commercial product, our *prototype of a prototyping environment* describes exemplarily the development of a "state-of-the-art"-CASE tool with a minimum of available means.

Due to the structure of TROLL specification documents (orthogonality of features for concepts) and the architecture of the *TBench* (aspects of integration) the results of the DFG project could be easily adapted to other application domains. Concrete next steps are mappings from TROLL to DDL's of existing database systems.

Acknowledgments. For many fruitful discussions on the *TBench* and TROLL we are grateful to our colleagues S. Conrad, G. Denker, R. Jungclaus, I. Schmitt, C. Tuerker and especially to H.-D. Ehrich as one of the leaders of the DFG-project.

References

[BKKZ92] R. Budde, K. Kautz, K. Kuhlenkamp, and H. Züllighoven. *Prototyping – An Approach to Evolutionary System Development*. Springer-Verlag, 1992.

[BM92] A. W. Brown and J. A. McDermid. Learning from IPSE's Mistakes. *Integrated Case, IEEE*, pages 23–28, March 1992.

[BP92] W. Bischofberger and G. Pomberger. *Prototyping-Oriented Software Development*. Springer, 1992.

[Bri93] S. Brinkkemper. Integration diagrams in case tools through modelling transparency. *Information and Software Technology*, 35(2):101–105, 2 1993.

[Cag90] M.R. Cagan. The HP Softbench environment: An architecture for a new generation of software tools. *In HP Journal 41*, pages 36–47, 1990.

[Con94] S. Conrad. *Ein Basiskalkül für die Verifikation von Eigenschaften synchron interagierender Objekte*. Fortschritt-Berichte Reihe 10, Nr. 295. VDI-Vieweg, 1994.

[Esp93] ESDI S.A, Lisbon. *OBLOG CASE V1.0 – User's Guide*, 1993.

[ESS92] H.-D. Ehrich, G. Saake, and A. Sernadas. Concepts of Object-Orientation. In *Proc. of the 2nd Workshop of "Informationssysteme und Künstliche Intelligenz: Modellierung", Ulm* , pages 1–19. Springer IFB 303, 1992.

[FSMS91] J. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-Theoretic Semantics of Object-Oriented Specification Constructs. In R. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction (Proc. 4th IFIP WG 2.6 Working Conference DS-4, Windermere)*, pages 243–284, Amsterdam, 1991. North-Holland.

[Gri82] J. J. van Griethuysen. Concepts and Terminology for the Conceptual Schema and the Information Base. Report N695, ISO/TC97/SC5, 1982.

[HJS93] T. Hartmann, R. Jungclaus, and G. Saake. Animation Support for a Conceptual Modelling Language. In V. Mařík, J. Lažanský, and R.R. Wagner, editors, *4th DEXA, Prague*, pages 56–67. LNCS 720, Springer, 1993.

[HSJ+94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch. Revised Version of the Modelling Language TROLL (Version 2.0). Informatik-Bericht 94–03, Technische Universität Braunschweig, 1994.

[JSHS95] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1995. *To appear.*

[JWH+94] R. Jungclaus, R.J. Wieringa, P. Hartel, G. Saake, and T. Hartmann. Combining TROLL with the Object Modeling Technique. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen. GI-Fachgespräch FG 1: Integration von semi-formalen und formalen Methoden für die Spezifikation von Software*, pages 35–42. Springer, Informatik aktuell, 1994.

[MBW80] J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong. A Language Facility for Designing Interactive Database-Intensive Applications. *ACM Transactions on Database Systems*, 5(2):185–207, 1980.

[MS92] P. Mi and W. Scacchi. Process Integration in CASE Environments. *IEEE Computer-Aided Software Engineering (CASE)*, pages 54–62, 1992.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, 1991.

[SB93] D. Schefström and G. van den Broek. *Tool Integration*. Wiley Professional Computing, 1993.

[SJH93] G. Saake, R. Jungclaus, and T. Hartmann. Application Modelling in Heterogeneous Environments using an Object Specification Language. *IJICIS*, 2(4):425–449, 1993.

[TN92] I. Thomas and B. A. Nejmeh. Definitions of Tool Integration for Environments. *IEEE Integrated Case*, pages 29–35, March 1992.

[WR92] H. Wächter and A. Reuter. The ConTract Model. In A. K Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan-Kaufmann, Palo Alto, 1992.