Publications of the DBIS Research Group

May 19, 1995

"Datenbanken und Informationssysteme (DBIS)"
Fachbereich Informatik, Universität Hamburg
Vogt-Kölln-Straße 30, D-22527 Hamburg, FRG
e-mail: matthes|... @informatik.uni-hamburg.de

DBIS research group:

Head of the group: Prof. Dr. Joachim W. Schmidt

DBIS members: Prof. Dr. Winfried Lamersdorf, Dr. Florian Matthes, Andreas Rudloff, Rainer Müller, Andreas Gawecki, Michael Merz, Bernd Mathiske, Kay Müller, Gerald Schröder, Sven Müßig

The list covers published papers and technical reports since 1990 and includes results on the following DBIS research topics:

- programming languages and database models, type systems, bulk types and iteration abstractions, integrated language and system development ...
- persistent object systems, object stores, store protocols, intermediate representations, program analysis, transformation and optimization, portability, interoperability and gateway development ...
- data-intensive applications in distributed environments, open systems and communication protocols, network management, federated systems, specification of services and interfaces, standards ...
- environments and methods for data-intensive applications, semantic modelling, typeful programming, object-oriented design, program generation and verification, systems construction and management ...

An online version of this document is available via NCSA Mosaic and the World-Wide-Web. Please connect to the following address:

`http://idom-www.informatik.uni-hamburg.de/Paper/entry.html`

The online document also contains hyper-links that provides direct access to PostScript versions of most of the documents. Further up-to-date information on DBIS research projects, publications in German and software prototypes is available through

`http://idom-www.informatik.uni-hamburg.de/external-entry.html`

1. Published Papers and Papers Submitted for Publication

[GaMa95] A. Gawecki and F. Matthes. Integrating Query and Program Optimization Using Persistent CPS Representations. (Submitted for publication.), February 1995.

To overcome the limitations resulting from the traditional separation between program and query optimization, the Tycoon[1] system integrates the analysis and rewriting of queries and programs using a common persistent intermediate representation (TML). TML is a continuation passing style (CPS) language which excels in its explicit, high-level representation of control and data dependencies. We describe Tycoon's innovative reflective system architecture that integrates program and query optimization and that supports modular compile-time as well as global run-time optimizations. We formally define TML and its core rewrite rules that unify many well-known optimizing transformations. Moreover, we describe how optimizations across abstaction barriers are performed in Tycoon and we report on our experience with optimizing large modular persistent applications.

[GaMa95a] A. Gawecki and F. Matthes. TooL: A Persistent Language Integrating Subtyping, Matching and Type Quantification. (Submitted for publication.), May 1995.

This paper presents the type system of the TooL persistent programming language that captures much of the flavor of Smalltalk within a safe static typing discipline. Following the spirit of Smalltalk providing a highly flexible and extensible programming environment based on a small set of expressive language primitives, TooL provides only few built-in type concepts with rich semantics that achieve power through systematic use and orthogonality. The paper focuses on the language design issues that arise in the course of integrating type concepts that are well-understood in isolation like object types, subtyping, type matching, and type quantification into a practical database programming language. We illustrate these issues by code samples taken from the TooL generic bulk data library.

[GGL+95] K. Geihs, H. Gründer, W. Lamersdorf, M. Merz, K. Müller, and A. Puder. Systemunterstützung für offene verteilte Dienstemärkte. In *Kommunikation in Verteilten Systemen: Neue Länder – Neue Netze – Neue Dienste*, Informatik-Aktuell, pages 445–459. Springer-Verlag, 1995.

Mit der globalen Vernetzung von Rechnersystemen entsteht ein wachsendes Potential für eine Vielzahl spezialisierter Online–Dienste, welche sowohl Endbenutzern als auch Software–Anwendungen zur Verfügung stehen. Derartige offene verteilte Dienstemärkte sind insbesondere durch ihre Vielfalt des Dienstangebotes und die Dynamik der dort stattfindenden Nachfrager-Anbieter-Beziehungen gekennzeichnet. Dieser Artikel analysiert die Anforderungen einer solchen Umgebung und schlägt ein Architekturmodell vor, welches eine adäquate Grundlage für die Spezifikation, die Vermittlung und die Benutzung solcher Dienste bietet. Insbesondere Systemdienste zur Vermittlung und zum Zugriff auf Dienstangebote bilden einen Schwerpunkt der Betrachtung. Dabei werden auch konkrete Lösungsansätze vorgestellt, die zur Zeit in den am vorliegenden Beitrag beteiligten Arbeitsgruppen verfolgt und prototypisch implementiert werden.

[MJML95a] K. Müller-Jones, M. Merz, and W. Lamersdorf. Kooperationsanwendungen: Integrierte Vorgangskontrolle und Dienstvermittlung in offenen verteilten Systemen. (akzeptiert fuer GI/SI-Jahrestagung in Zuerich, September 95).

Angesichts der Anzahl und Vielfalt vorhandener Standard– und Individualdienste in offenen verteilten Systemen gewinnt die Verwendung dieser Dienste innerhalb verteilter Anwendungen zunehmend an Bedeutung. Um die Dienste jedoch auch effizient

[1] Tycoon: Typed Communicating Objects in Open Environments.

in die Entwicklung komplexer verteilter Anwendungen einbeziehen zu können, sind geeignete Abstraktionen notwendig, die eine möglichst einfache Nutzung der Dienste ermöglichen. Hierdurch ergeben sich auch neue Anforderungen an die darunterliegende Systemtechnik, welche die notwendige Infrastruktur für eine solche Dienstnutzung bzw. Anwendungskooperation bereitstellen muß. Eine Analyse vorhandener Systeme zeigt jedoch, daß bisher keine integrierten Lösungsansätze existieren, welche die Aspekte der Dienstnutzung mit einer für kooperierende Anwendungen typischen vorgangsorientierten Verarbeitungssicht verbinden.
Auf dem Hintergrund dieser Einschränkungen stellt der vorliegende Beitrag die an der Universität Hamburg entwickelte TRADE–Architektur zur Unterstützung von Kooperationsanwendungen in offenen verteilten Systemen vor. Hierbei wird insbesondere auch auf die neu entwickelte Vorgangsbeschreibungssprache PAMELA eingegangen, welche eine Definition von Anwendungsvorgängen auf der hohen anwendungsspezifischen Abstraktionsebene der Kooperationsanwendungen bietet, in dem sie kontrollspezifische Aspekte der Vorgangsbearbeitung mit einer dienstorientierten Aktionsausführung verbindet.

[MJML95b] K. Müller-Jones, M. Merz, and W. Lamersdorf. The TRADEr: Integrating trading into DCE. In *Proceedings of the Third International Conference on Open Distributed Processing (ICODP '95), Brisbane, Australia*, February 1995.

Client support for locating, accessing, and using arbitrary services in open system environments emerges as one of the most interesting, complex, and practically relevant tasks of realising realistic open distributed systems applications. In the context of Open Distributed Processing (ODP), current standardisation efforts for a trading function play an increasingly important role for open system integration.
In addition to ongoing ODP standardisation activities official and de–facto standard system environments such as, e.g., the OSF Distributed Computing Environment (DCE), support efficient development and portability of distributed system applications. Therefore, time seems now ready to analyse and evaluate the use of such platforms also for developing efficient implementations of higher–level system support services, e.g., an ODP trading function.
This paper first elaborates on the specific potential of OSF DCE for supporting implementations of ODP trader functions. It then presents an architecture and reports on experiences with such an implementation in the context of system support for general service access, management and coordination in open distributed environments within the TRADE project. Finally, the paper draws the attention to still existing limitations and deficiencies of OSF DCE for realising ODP trader functions and proposes respective extensions to OSF DCE both at a conceptual and a concrete systems implementation level. According to such prototype implementation experiments, ODP trading functions can be integrated smoothly into a uniform standard system support platform (like DCE) and can be implemented efficiently by extending DCE by additional trading functions which specifically support service management, mediation and access for open distributed applications.

[MMS95a] B. Mathiske, F. Matthes, and J.W. Schmidt. On Migrating Threads. (Accepted for publication.), June 1995.

Based on the notion of persistent threads in Tycoon, we investigate thread migration as a programming construct for building activity-oriented distributed applications. We first show how astraight-forward extension of a higher-order persistent language can be used to define activities that span multiple (semi-) autonomous nodes in heterogeneous networks. In particular, we discuss the intricate binding issues that arise in systems where threads are first-class language citizens that may access local and remote, mobile and immobile resources.
We also describe how our system model can be understood as a promising generaliza-

tion of the more static architecture of first-order and higher-order distributed object systems. Finally, we give some insight into the implementation of persistent and migrating threads and we explain how to represent bindings to ubiquitous resources present at each node visited by a migrating thread on the network to avoid excessive communication or storage costs.

[MMS95b] B. Mathiske, F. Matthes, and J.W. Schmidt. Scaling Database Languages to Higher-Order Distributed Programming. (Submitted for publication.), March 1995.

We describe the Tycoon approach to scale the successful notion of a uniform, type-safe persistent object store to communication-intensive applications and applications where long-term activities are allowed to span multiple autonomous network sites. Exploiting stream-based data, code and thread exchange primitives we present several distributed programming idioms in Tycoon. These programming patterns range from client-server communication based on polymorphic higher-order remote procedure calls to migrating autonomous agents that are bound dynamically to networkresources present at individual network nodes. Following Tycoon's add-on approach, these idioms are not cast into built-in syntactic forms, but are expressed by characteristic programming patterns exploiting communication primitives encapsulated by library functions. Moreover, we present a novel form of binding support for ubiquitous resources which drastically reduces communication traffic for modular distributed applications.

[CMA94] L. Cardelli, F. Matthes, and M. Abadi. Extensible Grammars for Language Specialization. In C. Beeri, A. Ohori, and D.E. Shasha, editors, *Database Programming Languages (DBPL-4)*, Workshops in Computing. Springer-Verlag, February 1994.

A basic dilemma in the design of a database programming language is the choice between a user-friendly language with a rich set of tailored notations for schema definitions, query expressions, etc., and a small, conceptually simple core language. We address this dilemma by proposing extensible grammars, a syntax-definition formalism for incremental language extensions and restrictions based on an initial core language.
The translation of programs written in tailored object languages into programs of a core language is defined via syntax-directed patterns. In contrast to traditional macro-expansion and program-rewriting tools, our extensible grammars respect scoping rules. Therefore, it is possible to introduce new binding constructs like quantifiers, iterators, and type declarations, while avoiding problems with variable captures and name clashes.
We develop extensible grammars and illustrate their use by extending the lambda calculus with let-bindings, conditionals, and SQL-style query expressions. We then give a formal description of the underlying parsing, transformation, and substitution rules. We also highlight some implementation details of a type-safe, extensible parser package based on the concept of extensible grammars developed for the Tycoon programming environment.

[MaSc94] F. Matthes and J.W. Schmidt. Persistent Threads. In *Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB*, pages 403–414, Santiago, Chile, September 1994. (An extended version of this text appeared as [MaSc94b]).

Persistent threads are a database programming concept particularly well-suited for applications that manage long-term, distributed or cooperative activities. We introduce persistent threads as a novel form of bindings from data in persistent object stores to activated code and relate them to existing binding concepts found in database programming. We also describe the integration of persistent threads into a polymorphically-typed database language and its supporting layered system architecture with particular emphasis on abstractly-defined thread representations which support thread analysis, optimization and portability.

[MJM94] K. Müller, K. Jones, and M. Merz. Vermittlung und Verwaltung von Diensten in offenen verteilten Systemen. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen: Eine Herausforderung für die Informatik*, Informatik aktuell, pages 219–226, Hamburg, Germany, August 1994. Springer-Verlag.

In offenen verteilten Systemen wird zukünftig eine Vielzahl bekannter sowie unbekannter Dienste angeboten werden. Um diese Dienste effektiv und effizient nutzen zu können, sind adäquate Unterstützungsmechanismen für die Dienstvermittlung, die Dienstverwaltung, die Dienstkontrolle und den Zugriff auf die Dienste erforderlich. Zur Realisierung dieser Aufgaben stellen wir in dieser Arbeit die von uns entwickelte COSM/TRADE–Architektur vor, wobei der Schwerpunkt in der Betrachtung der Vermittlung und Verwaltung sogenannter klassifizierter Dienste durch den TRADE–Traders liegt.

[MML94] M. Merz, K. Müller, and W. Lamersdorf. Service Trading and Mediation in Distributed Computing Environments. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS '94)*, pages 440–457, Poznan, Poland, 1994. IEEE Computer Society Press.

The increased availability of global communication infrastructures allows providers and users of various application services to cooperate in nearly unlimited geographic scopes. Problems of heterogeneity and scale have motivated specific standardisation activities for client/server "trading" or service "mediation" components. Motivated by current limitations of the emerging ODP (Open Distributed Processing) trader, this paper argues for a broader concept of general service "mediation" as more appropriate for realistic open distributed environments. The proposed mediation concept addresses some of the heterogeneity and flexibility requirements of open service cooperation by a uniform "Service Interface Description Language" (SIDL). The goal is to support distributed application development for a "Common Open Service Market" (COSM) by facilitating flexible service selection and client/server interaction.
The paper also presents basic components of a generalised trading and mediation architecture as well as the status of a prototype implementation.

[MML94b] K. Müller, M. Merz, and W. Lamersdorf. Der TRADE-Trader: Ein Basisdienst offener verteilter Systeme. In C. Popien and B. Meyer, editors, *Neue Konzepte für die Offene Verteilte Verarbeitung*, volume 7 of *Aachener Beiträge zur Informatik*, pages 35–44. Verlag der Augustinus Buchhandlung, September 1994.

Dem Trader als wichtiger Basisdienst offener verteilter Systeme kommt zunehmend eine zentrale Bedeutung zu. Angesichts der Komplexität verteilter Systeme und der Vielzahl und Vielfältigkeit der angebotenen Dienste bietet der Trader adäquate Konzepte zur Strukturierung der Dienste an und stellt Mechanismen für die Dienstvermittlung und -verwaltung bereit. Dieser Artikel stellt die Funktionalität und Implementierung des TRADE-Traders vor, welcher im Rahmen des COSM/TRADE–Projektes an der Universität Hamburg prototypisch auf Basis des Distributed Computing Environment DCE realisiert ist. Am Beispiel der Implementierung des TRADE-Traders wird im vorliegenden Beitrag gezeigt, wie eine modulare Erweiterung bestehender verteilter Systemarchitekturen um Funktionen des Tradings und eine weitgehende Integration mit den bereits durch die Systemarchitektur bereitgestellten Diensten, insbesondere den vorhandenen Dienstvermittlungs- und Verwaltungsdiensten, erreicht werden kann.

[MRSS94] F. Matthes, A. Rudloff, J.W. Schmidt, and K. Subieta. A Gateway from DBPL to Ingres. In W. Litwin and T. Risch, editors, *Applications of Databases, First*

International Conference, ADB-94, volume 819 of *Lecture Notes in Computer Science*, pages 365–380, Vadstena, Sweden, June 1994. Springer-Verlag.

A gateway from DBPL (being a superset of Modula-2) to the commercial database system Ingres is described. DBPL extends Modula-2 by a new bulk data type constructor "relation", persistence, and high-level relational expressions (queries) based on the predicate calculus, thereby maintaining the basic concepts of the language like strong typing and orthogonality. The gateway enables the user to write "normal" DBPL programs for accessing Ingres databases. This is in contrast to typical implementations that embed SQL statements into a programming language and results in a fully transparent interface for DBPL programmers. DBPL queries and statements referring to Ingres tables are automatically converted into corresponding SQL statements, are evaluated by the Ingres database server and the results are transferred back under the control of the DBPL program. This procedure also resolves queries referring to both Ingres and DBPL tables. The design assumptions of the gateway and the used implementation methods are presented as well as design and implementation difficulties.

[RMS94] A. Rudloff, F. Matthes, and J.W. Schmidt. Security as an Add-On Quality in Persistent Object Systems. (Submitted for publication in the series: Workshops in Computing, Springer Verlag), 1994.

System security services like authentication, access control and auditing are becoming increasingly critical for information systems particularly in distributed heterogeneous environments. Since information system architectures are moving rapidly from centralized, *grand unifying architectures* towards open, service-oriented and communication-based environments ("Persistent Object Systems") constructed with well-organized component technologies it is essential that such structural changes are reflected adequately in the architecture of system security services.
In this paper we present an open, library-based approach to the security of Persistent Object Systems which generalizes and unifies the protection mechanisms that traditionally come bundled with database, communication or operating system services. More specifically, we illustrate how polymorphic typing can be exploited to abstract from particular commercially available security services, such as Kerberos, and how higher-order functions allow the user to add value to existing security services. Furthermore, we demonstrate how higher-order functions, first-class modules and reflection provide a technical framework for the realization of domain-specific security policies and for the systematic construction of security-enhanced activities.

[ScMa94] J.W. Schmidt and F. Matthes. The DBPL Project: Advances in Modular Database Programming. *Information Systems*, 19(2):121–140, 1994.

In the DBPL project we tackled the problem of supporting data-intensive applications in a single framework, clean and simple in its conceptual foundation and free of technical mismatches. Conceptually, we based the DBPL language on Modula-2 with three built-in extensions which, at that time, were considered necessary (and sufficient) for data-intensive applications:

- a parametric bulk type constructor for "keyed sets" or relations;
- a module concept which supports sharing across programs and implies persistence;
- a procedure concept with transactional semantics, i.e., serializability and recovery.

In implementing the DBPL system we adopted a rather strict approach by aiming for

- full orthogonality in our relationally extended type space;
- type-complete persistence, i.e. longevity of data from Booleans to relations;
- functional abstraction for relational expressions including recursion.

As a consequence, the DBPL project covers in a quite natural way approaches otherwise considered disjoint as, for example, complex objects, multibases and data deduction.
In this paper, besides reporting on project achievements and spin-offs, we also present our insight in good-quality project design and its dependency on conceptual simplicity and implementational strictness. This also includes early recognition of the essential project "terminator" which, in our project, was Modula-2's monomorphic type system and its intrinsic restrictions on systems extensibility.
The DBPL project results in a mature product for modular database application programming and a follow-up project based on polymorphic types and other higher-order concepts

[BMS93] A. Borgida, J. Mylopolous, and J.W. Schmidt. The TaxisDL Software Description Language. In M. Jarke, editor, *Database Application Engineering with DAIDA*, volume 1 of *Research Reports ESPRIT*, pages 65–84. Springer-Verlag, 1993.

The purpose of the TaxisDL language is to express the conceptual design of an information system. The focus of the design process includes the data classes of the proposed system, the functions and transactions manipulating them, and the larger conceptual groupings of these actions into long-term activities, which we call scripts. The design of the language is based on ideas from semantic data models and formal specification languages.

[CMAM93] R.C.H. Connor, R. Morrison, M.P. Atkinson, F. Matthes, and J.W. Schmidt. Programming in Persistent Higher-Order Languages. In *Proceedings of Euro-Arch'93 Congress*. Springer-Verlag, October 1993.

Persistent programming languages differ from traditional languages in that data of arbitrary lifetimes is fully governed by the type system. Such languages lead to radically different programming paradigms and methodologies for two important reasons: (1) the high-level type system may be relied upon as a protection mechanism even for long-term data; (2) data of any type, including first-class procedures and abstract data types, may be kept for arbitrary lifetimes. The combination of these means that the kind of sophisticated typing commonly used in programs which operate over short-term data may be extended to all data manipulated by a long-lived system. This paper exposes some of the ways in which well-known type system features may be powerfully used in contexts normally associated with operating systems and database management systems.

[GSW93] T. Günther, K.-D. Schewe, and I. Wetzel. On the Derivation of Executable Database Programs from Formal Specifications. In *Proceedings of the of Formal Methods Europe '93 Symposium - Industrial-Strength Formal Methods, Odense Teknikum, Denmark*, April 1993.

Achieving wide acceptance of formal methods in software development requires a smooth integration with requirements analysis, design and implementation. Especially for database application systems there exist well-known approaches to conceptual modeling as well as a sophisticated implementation technology on the basis of database programming languagues.
The work described in this paper is based on a scenario, where the B method is coupled with a conceptual modeling language TDL and a database programming language DBPL. Both these languages can be represented in B. We concentrate on the problem of characterizing those B specifications that are sufficiently refined in order

to be transformed into equivalent DBPL programs. This gives rise to some kind of implementability proof obligation.
Moreover, we show that the transformation itself can be regarded as a term rewriting task based on a representation by term algebras of the languages involved. For this task we exploit order-sorted algebra by using the OBJ system.

[MaSc93] F. Matthes and J.W. Schmidt. DBPL: The System and its Environment. In M. Jarke, editor, *Database Application Engineering with DAIDA*, volume 1 of *Research Reports ESPRIT*, pages 319–348. Springer-Verlag, 1993.

The DBPL system and its environment support the modular implementation of advanced data-intensive applications based on integrated database and programming language technology. It provides state-of-the-art system support at application compile-, link- and run-time as well as tools for incremental application evolution.

[MaSc93a] F. Matthes and J.W. Schmidt. System Construction in the Tycoon Environment: Architectures, Interfaces and Gateways. In *Proceedings of Euro-Arch'93 Congress*. Springer-Verlag, October 1993.

This paper outlines the basic concepts and the system architecture of the Tycoon environment. Tycoon is designed for the construction of persistent object systems intended to be available on multiple software and hardware platforms.
Tycoon's contribution to system portability and scalability is achieved by strictly separating concerns of data modelling, data manipulation, and data storage into three distinct system layers each of which is based on state-of-the-art system technology (polymorphic programming languages, portable code representations, persistent object stores). Much emphasis is being placed on supporting interoperability between Tycoon applications and off-the-shelf tools and systems, such as database systems, user interface managers, and optimizing code generators.
We illustrate how higher-order functions, polymorphic typing and transparent persistence management reduce the amount of repetitive and type-unsafe programming in typical persistent object systems.

[Matt93] F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer-Verlag, 1993.

Unter dem Begriff der Persistenten Objektsysteme werden komplexe Anwendungen zusammengefaßt, die aufgrund ihrer langen Lebensdauer, ihrer Nähe zu Endbenutzern und ihrer Sicherheitsanforderungen auf ein hohes Maß an Unterstützung durch externe Diensterbringer angewiesen sind. Beispiele kommerzieller Diensterbringer sind Objektspeicher, Transaktionsmonitore, Datenbanksysteme, Generatoren für graphische Benutzerschnittstellen oder generische Dienste zur Kommunikation zwischen verteilten Objekten. Bislang sind derartige Diensterbringer weitgehend unabhängig voneinander entwickelt worden und können nur über historisch gewachsene, schmale Schnittstellen miteinander kommunizieren.
Die Qualität zukünftiger persistenter Objektsysteme wird jedoch entscheidend durch die Flexibilität, Effizienz und Korrektheit im Zusammenspiel der Objekte dieser verschiedenartigen generischen Dienste und weniger durch die Leistungsfähigkeit einzelner isolierter Systemkomponenten bestimmt.
In diesem Buch wird im Detail die persistente polymorphe Programmierumgebung Tycoon (Typed communicating objects in open environments) beschrieben, die eine Verbesserung der Qualität persistenter Objektsysteme dadurch erreicht, daß der Anteil der stereotypen, untypisierten Programmiertätigkeiten erheblich reduziert werden kann (verglichen etwa mit C, Modula-2 oder C++ Lösungen).
Konkret bietet das Tycoon System spezielle sprachliche und architekturelle Unterstützung für die vollständig transparente Verwaltung langlebiger Daten- und Programmobjekte, die mengenorientierte Verarbeitung (benutzerdefinierter) Kollektio-

nen, die Implementierung generischer Bibliotheken und die typsichere Anbindung externer Systemsoftware (z.B. SQL Datenbanken oder Fenstersysteme).
Das Buch beschreibt die zentralen Komponenten des Tycoon Systems nicht nur aus der Sicht des programmierenden Anwenders, sondern vermittelt auch Einblicke in die interne Systemarchitektur sowie die Algorithmen und Formalismen zur Analyse und Übersetzung generischer persistenter Programme. Schließlich enthält der Text substantielle Beispiele für die Konstruktion generischer Bibliotheken und Beispiele objekt-orientierter, funktionaler und relationaler Programmiertechniken in Tycoon.

[MeLa93] M. Merz and W. Lamersdorf. Generic Interfaces to Remote Applications in Open Systems. In *Proceedings of the International IFIP Workshop on Interfaces in Industrial Production and Engineering Systems*, pages 267–281. North–Holland, Elsevier Science Publishers B.V., March 1993.

Future industrial production and engineering environments will profit substantially from emerging open distributed computer communication network environments. They will, in principle, be able to benefit from a high potential of services available in such environments to support individual client applications. In practice, however, free and flexible client/server cooperations are frequently hindered by the great and confusing variety of interfaces involved in accessing various and heterogeneous network services. In order to support open client/server cooperations in distributed systems, this contribution proposes a unifying description mechanism for remote services in computer networks. It describes an application oriented generic communication service, which facilitates client/server cooperation in open systems. The most important basis for such a service is a uniform service specification mechanism for open server interfaces. Correspondingly, the paper first specifies a specific service interface description language (SIDL). It then shows how such a service interface description could also be used for automatic creation of server-specific local human user interfaces. In combination, a generic network interface description language (NIDL) specification, as proposed here, supports client applications in open systems by providing a common mechanism to access and utilize any service available anywhere in the network.

[ScMa93] J.W. Schmidt and F. Matthes. Lean Languages and Models: Towards an Interoperable Kernel for Persistent Object Systems. In *Proceedings of the IEEE International Workshop on Research Issues in Data Engineering RIDE '93 Vienna, Austria*, pages 2–17, April 1993. (Also appeared as TR FIDE/93/59).

Reliable interoperation between independently developed systems frequently requires type-safe access to persistent data objects and generic services while today's system architectures and interoperation tools still focus primarily on store-level access to volatile data and simple monomorphic or untyped services.
In this paper, we summarize our experience gained in a long-term project that provides persistence abstractions and generic database support in a strongly typed database environment which includes optimizing gateways to commercial relational database servers and module-based distributed programming tools. To keep the presentation focussed, we make use of a uniform language model based on higher-order polymorphic types to capture the essential interoperation semantics including classical cross-language calling mechanisms, remote procedure call models as well as relational and object-based database gateways.
This uniform language model is also the conceptual core of the Tycoon database environment being developed at Hamburg University. Tycoon lifts Persistent Object System interoperability to a higher level of genericity and precision while further reducing overall system complexity by a lean approach to languages and models for data, execution and storage.
Since it is central to our concept of lean software production to substantially reduce

the depth of manufacturing penetration by importing and reusing external services, interoperability is crucial to our approach.

[ScMa93a] J.W. Schmidt and F. Matthes. Modular and Rule-Based Database Programming in DBPL. In M. Jarke, editor, *Database Application Engineering with DAIDA*, volume 1 of *Research Reports ESPRIT*, pages 85–122. Springer-Verlag, 1993. (Also available as TR FIDE/91/15).

DBPL is the target language for the implementation of data-intensive applications in the DAIDA project. This paper illustrates how DBPL supports the construction of correct, modular and efficient database programs integrating well-developed compilation and database technology.
After an introduction to the language concepts of DBPL, two methodoligical aspects central to the implementation of database applications are discussed. The first task is the representation of the classical abstraction principles of Conceptual Modelling (classification, aggregation, generalization, and association) by typed data structures and data derivation rules in DBPL. The second aspect is the decomposition of application systems consisting of multiple interacting programs into managable, shared modules using a novel abstraction mechanism for bulk data access.
By presenting the use of DBPL in some detail, we intend to support our claim that a set- and predicate-oriented view of database modelling resolves the traditional mismatch between declarative database languages and their general-purpose imperative counterparts.

[SMRS93] K. Subieta, F. Matthes, A. Rudloff, J.W. Schmidt, and I. Wetzel. Viewers: A Data-World Analogue of Procedure Calls. In *Proceedings of 19th Intl. Conf. on Very Large Data Bases VLDB '93, Dublin, Ireland*, August 1993.

A viewer is a reference-valued datum with a special meaning: a value of the data pointed by the viewer becomes a virtual part of data where the viewer is placed; the value virtually substitutes the viewer. Viewers are considered to be a data-world analogue of procedure calls. They possess a large conceptual and pragmatic potential as a result of new data semantics on which we can base a variety of well-organized data structures.
Various applications of viewers, related to DBPLs and object-oriented data modelling, are presented: importing common attributes, inheritance and multi-inheritance, stored selections, projections and joins, viewing a single relational structure as several hierarchical structures, etc. Methodological and formal aspects of the concept are discussed and a method of incorporating viewers into a query language is presented.

[SMV93] J.W. Schmidt, F. Matthes, and P. Valduriez. Building Persistent Application Systems in Fully Integrated Data Environments: Modularization, Abstraction and Interoperability. In *Proceedings of Euro-Arch'93 Congress*. Springer-Verlag, October 1993.

Research and development in the FIDE project on Fully Integrated Data Environments has led to the concept of Persistent Object Systems (POS) which generalize database systems by re-interpreting schemas as type definitions and databases as typed variables in addition to treating lifetime as a type-independent property. Furthermore, FIDE develops uniform linguistic interfaces for data modelling, computation and communication, and extends database, programming and communication technology to enable integrated application development based on interoperating POSs.
As a consequence of such an integrated view, formerly disjoint concepts such as databases, program and module libraries, files or repositories can now be treated uniformly as POSs differentiated essentially by the types of objects they contain and by the operational abstractions they provide.
Based on state-of-the-art database technology, this paper outlines FIDE results in

extending databases and providing the integrated technology considered necessary for the construction and maintenance of Persistent Application Systems. Since our main goal is to improve substantially a system's capability of persisting successfully over time in changing environments, particular emphasis will be placed on systems scalability and its consequences for POS interoperability.

[STW93] K.-D. Schewe, B. Thalheim, and I. Wetzel. Integrity Preserving Updates in Object-Oriented Databases. In *Proceedings of the Fourth Australian Database Conference ADC '93, Brisbane Queensland, Australia*, February 1993.

Integrity with respect to static and dynamic constraints is a cruicial property of database application systems, especially in OODBs, where structurally defined constraints such as inclusion and referential constraints have to be preserved. In this paper we solve this problem for basic update operations that are determined by the schema. We present a framework on rewriting of generic operations in dependence on the structural and semantical information of the schema. We show how consistency-preserving, basic update operations can be generated. Moreover, we analyse distinguished classes of user-defined constraints and generalize our results to them.

[WSS93] I. Wetzel, K.-D. Schewe, and J.W. Schmidt. Specification and Refinement of Databases. In M. Jarke, editor, *Database Application Engineering with DAIDA*, volume 1 of *Research Reports ESPRIT*, pages 283–318. Springer-Verlag, 1993.

Traditionally, substantial portion of database application semantics are captured through static and dynamic integrity constraints. The work reported in this paper exploits this fact by interpreting such database constraints as invariants and pre- and postconditions in the style of 'Z'.
Database applications are specified by a conceptual modelling language close to TAXIS, which has been enriched by constructs for a predicative specification style. Conceptual designs of database applications are formally analyzed for consistency and are refined, step by step, into efficient extended relational implementations. The reification of designs into implementations uses the formal framework of Abrial's Abstract Machines and generalized substitutions.
It is shown that a small set of standard refinement rules is sufficient for a wide class of refinements. Furthermore, it is argued that the proposed proof-based approach has significant advantages over the traditional database technique, which tests constraints expensively at transaction commit time.

[PSS92] B. Piza, K.-D. Schewe, and J.W. Schmidt. Term Subsumption with Type Constructors. In *Proceedings of the 1st Int. Conf. on Data and Knowledge Management, Baltimore*, May 1992.

Term Subsumption Languages (TSLs), a generalization of both semantic networks and frames equipped with a model-theoretic semantics, have shown to be of vital practicability for the representation of knowledge. Recently it has been claimed that the modelling power of TSLs could be decisively enhanced by introducing type constructors as they are used in modern typed programming languages.
In this paper we do a first step into this direction regarding only those type constructors that are commonly used in semantic data models (SDMs). By exploiting the similarity of abstraction mechanisms in SDMs and TSLs we define the integrated representation model IFO+ and demonstrate that the inference mechanisms of both approaches can also be unified. We show in particular that subsumption can be regarded as a stronger version of absolute dominance.

[SSW92] K.-D. Schewe, J.W. Schmidt, and I. Wetzel. Identification, Genericity and Consistency in Object-Oriented Databases. In J. Biskup and R. Hull, editors, *Database Theory – ICDT '92*, volume 646 of *Lecture Notes in Computer Science*, pages 341–356. Springer-Verlag, October 1992.

It is claimed that object-oriented databases overcome many of the limitations of the relational data model especially by generalizing the notion of object identification. A clear distinction between objects and values turns out to be essential for the object-oriented approach whereas the relational model is based exclusively on values. Since, however, value uniqueness within scopes is a quite natural constraint for a wide class of applications, identification by value is also of interest for object-oriented datamodels.
Hence, in this paper we concentrate on those classes where the extents are completely representable by values. We formalize some basic concepts of object-oriented databases and show that the finiteness of a database and the existence of finitely representable recursive types are sufficient to decide value-representability.
Another advantage of the relational approach is the existence of structurally determined canonical update operations. We show that this property can be carried over to object-oriented datamodels iff classes are value-representable. Moreover, in this case database consistency with respect to implicitly specified referential and inclusion constraints will be automatically preserved.

[STSW92] K.-D. Schewe, B. Thalheim, J.W. Schmidt, and I. Wetzel. Integrity Enforcement in Object-Oriented Databases. In U. Lipeck, editor, *Proc. 4th Int. Workshop on Foundations of Models and Languages for Data and Objects, Volkse, Germany, October 19–22*, 1992.

In contrast to the relational model methods in OODBs must enforce structurally defined constraints such as inclusion and referential constraints. It has been shown that this is possible for basic generic update operations that are determined by the schema. However, such operations only exist for value-representable classes.
In this paper we generalize this result and show that integrity enforcement is always possible. Given some arbitrary method S and some static or transition constraint I there exists a greatest consistent specialization (GCS) S(I) of S with respect to I. Such a GCS behaves nice in that it is compatible with the conjunction of constraints, inheritance and refinement.
Moreover, it is possible to derive simple representations of GCSs for basic update operations with respect to distinguished classes of explicitly stated static constraints. For the GCS construction of a user-defined operation, however, it is in general not sufficient to replace the involved primitive update operations by their GCSs.

[ERMS91] J. Eder, A. Rudloff, F. Matthes, and J.W. Schmidt. Data Construction with Recursive Set Expressions. In *Proceedings of the Kiev East/West Workshop on Next Generation Database Technology*, volume 504 of *Lecture Notes in Computer Science*, April 1991. (Also appeared as TR FIDE/91/13).

In this paper we present a conceptually rather conservative approach to data deduction. Instead of introducing new language constructs we stay within the conventional relational framework, exploit it, however, further by making better use of current language technology. Applying naming, typing and binding to queries and relations leads to a language that gains expressiveness from its orthogonality rather than from extensiveness.
As a framework for our presentation we use DBPL, a strongly typed database programming language based on the relational calculus and on Modula-2. In DBPL, data construction is expressed declaratively through set-oriented expressions which can be abstracted by parameterization and by naming, thus allowing powerful recursive constructor definitions. In the paper, a model-theoretic constructor semantics is defined in two steps: parameter substitution in constructor definitions leads to constructor instances which are then evaluated in a second step. The first step can be regarded as logic program generation, the second step as program evaluation.
We show, that for the general case no procedure exists that evaluates an arbitrary

set of parameterized constructors and is guarantied to terminate. However, we are able to classify constructors and give an evaluation algorithm which terminates for interesting subclasses. Finally, we solve an example from the literature showing that np-complete problems can be solved by constructors from a subclass which is decidable. This implies that decidable DBPL constructors are strictly more expressive than stratified Datalog.

[JMSV91a] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou. Information Systems Development as Knowledge Engineering: A Review of the DAIDA Project. *Programmirowanie, Software Journal*, 1, January 1991.

The development of database-intensive information systems has to consider a rich set of constraints from many different viewpoints: faithful representation of the application domain, good fit into the usage environment, maintainable conceptual organization of a large software system, cost-saving through reuse of existing experiences, efficient handling of massive data amounts, etc. To unify these viewpoints, we propose to treat them formally as interrelated knowledge bases, and to manage their evolution as a knowledge engineering process. The process is guided by design decisions within the constraints of a formal environment model. Experiences in ESPRIT project DAIDA demonstrate advantages of this approach in terms of conceptual flexibility, process support, and quality assurance.

[MaSc91] F. Matthes and J.W. Schmidt. Towards Database Application Systems: Types, Kinds and Other Open Invitations. In *Proceedings of the Kiev East/West Workshop on Next Generation Database Technology*, volume 504 of *Lecture Notes in Computer Science*, April 1991. (Also appeared as TR FIDE/91/14).

The activity of software development is moving rapidly from coding of programs to modelling of systems by means of services provided by open and modular environments. This shift enables service suppliers to gain customers by generalizing the functionality of their products, and it allows service consumers to conveniently buy functionality by just specializing their needs. It also motivates consumers to construct applications with more than just the most specialized functionality in mind thus contributing to system extensibility and reusability. Finally, the novel view on software construction blurs the formerly quite sharp distinction between the application part and the system part of software development.
In this paper we apply the consequences of the process sketched above to database systems and data-intensive applications. We argue in favor of a more open but yet controlled interaction between database sytems and their applications and we discuss the implications on the major abstraction principles to be supported by future database programming languages. Finally we follow the open invitation issued by novel computer languages to exploit their potent conceptual basis for the benefit of next generation database application systems.

[MaSc91a] F. Matthes and J.W. Schmidt. Bulk Types: Built-In or Add-On? In *Proceedings of the Third International Workshop on Database Programming Languages, Nafplion, Greece*. Morgan Kaufmann Publishers, September 1991. (Also appeared as TR FIDE/91/27).

Bulk structures play a central role in data-intensive application programming. The issues of bulk type definition and implementation as well as their integration into database programming languages are, therefore, key topics in current DBPL research.
In this paper we raise a more general language design and implementation issue by asking whether there should be at all built-in bulk types in DBPLs. Instead, one could argue that bulk types should be realized exclusively as user-definable add-ons to unbiased core languages with appropriate primitives and abstraction facilities.
In searching for an answer we first distinguish two substantially different levels on

which bulk types are supported. Elementary Bulk essentially copes with persistent storage of mass data, their identification and update. Advanced Bulk provides additional support required for data-intensive applications such as optimized associative queries and integrity support under concurrency and failure.
Our long-term experience with bulk types in the DBPL language and system clearly shows the limitation of the built-in approach: built-in Advanced Bulk, as elaborate as it may be, frequently does not cover the whole range of demands of a fully-fledged application and often does not provide a decent pay-off for its implementation effort. On the other hand, restriction to built-in Elementary Bulk gives too little user-support for most data-intensive applications.
We report our current work on open database application systems which favours DBPLs with bulk types as add-ons, and outline some of the technological requirements for highly reusable implementations of languages with advanced user-provided bulk type definitions.

[MOS91] F. Matthes, A. Ohori, and J.W. Schmidt. Typing Schemes for Objects with Locality. In *Proceedings of the Kiev East/West Workshop on Next Generation Database Technology*, volume 504 of *Lecture Notes in Computer Science*, April 1991. (Also appeared as TR FIDE/91/12).

The crucial characteristic of object-oriented databases is the concept of object identity which allows the direct representation of various kinds of dependencies between objects, for example, sharing and cyclicity. For object stores to become a viable technology for large shared databases, a certain degree of spatial control over object dependencies (or object locality) seems to be essential. This paper exploits the power of a static type system to capture and evaluate locality information on objects. First, we represent objects by references to complex expressions in a functional language. To control the locality of objects, the space of references is partitioned into a set of subspaces with an explicit reachability constraint. Next, we define a type system where the locality of an object is part of its static type specification and the predefined reachability constraint is enforced via a static typing discipline. We conclude by highlighting the impact of locality information on the operational support to be expected by next generation database systems.

[ScMa91] J.W. Schmidt and F. Matthes. The Rationale behind DBPL. In *3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems*, volume 495 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1991.

The DBPL language orthogonally integrates sets and first-order predicates into a strongly and statically typed programming language, and the DBPL system supports the language with full database functionality including persistence, query optimization and transaction management. The application of modern language technology to database concepts results in new insights into the relationship between types and schemas, expressions/iterators and queries, selectors and views, or functions and transactions. Furthermore, it allows the exploitation of type theory and formal semantics of programming languages and thus connects database application development with results from program specification and verification.

[ScMa91a] J.W. Schmidt and F. Matthes. Relational Database Programming: Naming, Typing and Binding. In *Proceedings of the Programming School'91, Bulgaria*, February 1991.

The DBPL language orthogonally integrates sets and first-order predicates into a strongly and statically typed programming language, and the DBPL system supports the language with full database functionality including persistence, query optimization and transaction management. Modern language technology with its sound naming, typing and binding schemes applied to database concepts results in new insights into the relationship between types and schemas, expressions/iterators and queries, selectors and views, or functions and transactions.

[JMWJ90] M. Jeusfeld, M. Mertikas, I. Wetzel, M. Jarke, and J.W. Schmidt. Database Application Development as an Object Modelling Activity. In *Proc. 16th VLDB Conference*, Brisbane, Australia, August 1990.

The DAIDA project has made an attempt of formally defining and relating declarative and computational entities considered relevant to the process of database application development. Such entities range from object-oriented specifications to executable modules of database programs. To bridge the gap between semantics and computation, they also include abstract machine-based formal specifications and transformational theories. In a second contribution, selected characteristics of such entities and relationships are modeled uniformly in a software information system. Emphasis is placed on those properties that may become relevant when applications have to be modified or adjusted. Besides discussing the interaction of these aspects of the DAIDA methodology, the paper outlines an operational project prototype and reports first experiences.

[ScMa90] J.W. Schmidt and F. Matthes. Naming Schemes and Name Space Management in the DBPL Persistent Storage System. In *Persistent Object Systems*, Martha's Vineyard, Massachusetts, USA, September 1990. Morgan Kaufmann Publishers, Inc.

Database applications must be capable of dynamically creating large quantities of names based on a scheme that can be communicated to and shared by some user community. The paper concentrates on this particular aspect of the set- and predicate-based Data Base Programming Language DBPL. It re-interprets the key and view mechanisms of relational database systems as naming schemes by introducing the concept of a selector for user-defined associative naming of partitions of large data sets. The DBPL approach results in a novel interpretation of relations as scopes with variable extents of named and typed data objects. The consequences of associative naming schemes on typing and binding mechanisms are discussed and various extensions required by data-intensive applications are introduced. The paper also describes the system support that the DBPL implementation provides for efficient name space management.

[ScMa90a] J.W. Schmidt and F. Matthes. Language Technology for Post-Relational Data Systems. In *Database Systems of the 90s*, volume 466 of *Lecture Notes in Computer Science*, Berlin, Germany, November 1990. Springer-Verlag.

Practice has proven that databases are the keystones for nearly all application systems with a wider functionality, utilization and availabilty. As a consequence, next generation database systems will have to provide their services with a degree of interoperability that has to be substantially improved over existing solutions. In this paper we argue that this objective can be achieved only through full exploitation of current developments in computer language technology.
We claim that the merits of modern computer languages' naming schemes, typing systems and binding mechanisms are vital also for database management systems if they want to improve the quality of their interaction with application languages and programming environments as well as their own functionality and extensibility.
This paper studies the technological basis of modern computer languages and presents DBPL as a working example of a highly interoperable database programming language that exploits such technology.
Requirements of future data systems are discussed by emphasizing the abstraction principles considered helpful for the adequate design and organization of data-intensive applications and for the modularization, localization and, finally, the implementation of data-based systems. We conclude by relating the potential of advanced

language technology to such specific demands of the next generation of post-relational data systems.
Key words: database programming languages, query languages, interoperability, open systems, language design principles, naming schemes, type systems, binding mechanisms, DBPL.

[MaSc89] F. Matthes and J.W. Schmidt. The Type System of DBPL. In *Proc. of the 2nd Workshop on Database Programming Languages, Salishan Lodge, Oregon*, pages 255–260, June 1989.

This paper, compairing certain critical points of the type system design, presents the type system of the database programming language DBPL within a framework proposed by Albano et al. in the same volume.

[ScMa89] J.W. Schmidt and F. Matthes. Advances in Database Programming: On Concepts, Languages and Methodologies. In *Proc. 16th SOFSEM'89*, Zdiar, High Tatra, CSSR, December 1989. Available through Hamburg University.

The development of data-intensive applications, such as information systems or design applications in mechanical or software engineering, involves requirements modelling, design specification, and, last but not least, the implementation and maintenance of large database application programs. This paper concentrates on the last issue and presents the state-of-the art in Database Programming.
Over the years, computer science has isolated a number of requirements essential for data modelling and has developed a variety of solutions on those demands. Examples include the various concepts to identify, structure and constrain data, as well as contributions to data visibility, lifetime, abstraction etc.
The paper first characterizes data-intensive applications by their specific demands in data modeling and argues for database programming languages and environments based on highly integrated and consistent sets of data modelling concepts. Next, the classification scheme for Database Programming Languages (DBPLs) is introduced and applied to several DBPLs. The paper concludes with initial results on database programming methodologies and environments that support the engineering of data-intensive applications.

[SGLJ89] J.W Schmidt, L. Ge, V. Linnemann, and M. Jarke. Integrated Fact and Rule Management Based on Database Technology. In J.W. Schmidt and C. Thanos, editors, *Foundations of Knowledge Base Management*, Topics in Information Systems. Springer-Verlag, 1989.

Database programming languages integrate concepts of databases and programming languages to provide both implementation tools for data-intensive applications and high-level user interfaces to databases. Frequently, database programs contain a large amount of application knowledge which is hidden in the procedural code and thus difficult to maintain with changing data and user views.
This chapter presents a first attempt to improve the situation by supporting the integrated definition and management of data and rules based on a set-oriented and predicative approach. For the definition of rules, we introduce and justify a new declarative language called constructor. Furthermore, we demonstrate how a Recursive Data Model can be used for constructor representation, thus allowing for the definition, update, and querying of large rule bases. The use of database technology for integrated fact and rule base management is shown to have some important advantages in terms of fact and rule integrity, question-answering, and explanation of results.

[SWBM89] J.W. Schmidt, I. Wetzel, A. Borgida, and J. Mylopoulos. Database Programming by Formal Refinement of Conceptual Designs. *IEEE – Data Engineering*, September 1989.

In the process of developing an Information System, one passes through stages that include requirements gathering, design specification, and software implementation. The purpose of the TDL language is to express the conceptual design of an information system; it is the intermediate language in a triad that includes the language Telos, which captures an evolutionary view of the application domain and requirements, and DBPL, a procedural programming language that has persistent values and transactions supporting the development of databases. We consider the TDL's features for specifying the data and eventual procedural components of the system, and discuss how these are related to its companions. We also survey several tools for manipulating TDL descriptions which are currently under development, and give a detailed example of the iterative refinement of TDL designs into DBPL programs.

2. Technical Reports

[CMA94a] L. Cardelli, F. Matthes, and M. Abadi. Extensible Syntax with Lexical Scoping. Technical Report 121, Digital Equipment Corporation, Systems Research Center, Palo-Alto, California, February 1994.

A frequent dilemma in programming language design is the choice between a language with a rich set of notations and a small, simple core language. We address this dilemma by proposing extensible grammars, a syntax-definition formalism for incremental language extensions and restrictions.
The translation of programs written in rich object languages into a small core language is defined via syntax-directed patterns. In contrast to macro-expansion and program-rewriting tools, our extensible grammars respect scoping rules. Therefore, we can introduce binding constructs while avoiding problems with unwanted name clashes.
We develop extensible grammars and illustrate their use by extending the lambda calculus with let-bindings, conditionals, and constructs from database programming languages, such as SQL query expressions. We then give a formal description of the underlying rules for parsing, transformation, and substitution. Finally, we sketch how these rules are exploited in the implementation of a generic, extensible parser package.
This text is a revised and expanded version of [CMA94]

[GaMa94] A. Gawecki and F. Matthes. The Tycoon Machine Language TML: An Optimizable Persistent Program Representation. FIDE Technical Report FIDE/94/100, Fachbereich Informatik, Universität Hamburg, Germany, August 1994.

This document provides a brief overview of the Tycoon Machine Language TML which is used as a persistent intermediate program representation within the Tycoon system. TML representations of Tycoon programs provide the basis for extensive tree analysis and rewriting aiming at various optimizations which are independent of source languages and target machines.
TML representations are used for both static (compile-time) and dynamic (runtime) optimizations. Dynamic optimizations are of particular importance for performance improvement of large and modular programs.
TML is based on Continuation Passing Style (CPS), a powerful yet simple program representation technique developed as a framework for optimizing compilers. The advantage of CPS lies in the significant reduction in number of program constructs to be handled by the Tycoon static and dynamic (reflective) optimizer.
We propose that in modern persistent data-intensive applications traditional database query optimization be integrated into a more generalized framework for extensive program analysis and transformation.

[MaSc94b] F. Matthes and J.W. Schmidt. Persistent Threads. FIDE Technical Report FIDE/94/88, Fachbereich Informatik, Universität Hamburg, Germany, August 1994. (A shorter version of this text appeared as [MaSc94].).

Persistent threads are a database programming concept particularly well-suited for applications that manage long-term, distributed or cooperative activities. We introduce persistent threads as a novel form of bindings from data in persistent object stores to activated code and relate them to existing binding concepts found in database programming. We also describe the integration of persistent threads into a polymorphically-typed database language and its supporting layered system architecture with particular emphasis on abstractly-defined thread representations which support thread analysis, optimization and portability.

[MMS94] F. Matthes, S. Müßig, and J.W Schmidt. Persistent Polymorphic Programming in Tycoon: An Introduction. FIDE Technical Report FIDE/94/106, Fachbereich Informatik, Universität Hamburg, Germany, August 1994.

This text provides an introduction to Tycoon (Typed Communicating Objects in Open Environments), an open persistent polymorphic programming environment. The Tycoon language TL is based on expressive and orthogonal naming, typing and binding concepts as they are required, for example, in advanced data-intensive applications. The characteristic language mechanisms of TL are first-class functions and modules, parametric and subtype polymorphism extended to a fully higher-order type system. Tycoon programs are statically typed but may include explicit dynamic type variables which can be inspected at run-time.

[SBMS94] K. Subieta, C. Beeri, F. Matthes, and J.W. Schmidt. A Stack-Based Approach to Query Languages. (Submitted for publication, available through DBIS, Fachbereich Informatik, Universität Hamburg), 1994.

Scoping, naming and binding are central concepts in the definition and understanding of programming languages. With the introduction of sophisticated data models, these issues become important for query languages as well. Additionally, the goal of integrating query and programming languages requires a common basis for their operational semantics. We offer here an approach to the definition of the operational semantics of query languages based on an abstract machine, in which names, their bindings, and scopes defined by query and data structure are central. The machine has own simple data model for its store, and has a stack for dealing with scopes. We argue for the generality of the approach and illustrate it by defining the semantics of many query language primitives. Finally, we briefly consider how assignment and procedures can be neatly added.

[MaSc93b] F. Matthes and J.W. Schmidt. Datenbankprogrammiersprachen. (Unpublished article, available through DBIS, Fachbereich Informatik, Universität Hamburg), 1994.

Datenbankprogrammiersprachen entstehen durch die konzeptuelle Integration von Datenbankmodellen und algorithmisch vollständigen Programmiersprachen. Sie leisten damit einen Beitrag zur Realisierung adäquater Entwicklungsumgebungen für datenintensive Anwendungen.
Motiviert wird diese Integration einerseits durch die inzwischen allgemein klar erkannte Diskrepanz zwischen mengenorientierter, deklarativer Datenverarbeitung in Datenbanksystemen und dem elementorientierten, prozeduralen Paradigma konventioneller Programmiersprachen (impedance mismatch). Darüber hinaus gibt es jedoch noch weitere Unverträglichkeiten wesentlicher Konzepte in Datenbankmodellen und Sprachen zur Anwendungsprogrammierung; Beispiele hierfür sind Benennbarkeits- und Strukturierungsmechanismen sowie Bindungszeitpunkte und Methoden zur Lebensdauerkontrolle (competence mismatch). Diese Inkompatibilitäten führen nicht nur zu oft beklagten Performanzverlusten an der Schnittstelle zwischen Wirtsprache und Datenbanksystem, sondern erzwingen vor allem redundante Deklarationen und aufwendige repetitive Konvertierungsoperationen auf niedrigem Abstraktionsniveau, die hohe Folgekosten im gesamten Lebenszyklus der Applikationsprogramme nach sich ziehen.

[MeLa93a] M. Merz and W. Lamersdorf. Schnittstellenspezifikation in offenen Systemen. In *Proceedings of GI/ITG Arbeitstreffen: Entwicklung und Management verteilter Anwendungssysteme*, Frankfurt/Main, Germany, October 1993.

In den immer größeren und immer weiter verbreiteten, offenen verteilten Systemumgebungen steht im Prinzip auch eine immer größere Zahl von Diensten zur Unterstützung der Kooperation bei verteilten Anwendungen zur Verfügung. Deren praktische Verwendungsmöglichkeit ist in derartigen Umgebungen jedoch häufig eingeschränkt, da

der Zugang zu diesen Diensten ohne Kenntnis sowohl ihrer Existenz als auch ihrer jeweiligen Schnittstelle undurchführbar wird. Deshalb sollte eine Infrastruktur zur Unterstützung der Entwicklung verteilter Anwendungen in einem solchen 'offenen Markt von Diensten' [MeLa93b] u.a. auch Mechanismen zur Beschreibung von entfernten Anwendungsdiensten und ihren jeweiligen Schnittstellen vorsehen.
Im vorgelegten Diskussionsbeitrag wird dazu zunächst ein eigener Ansatz einer Schnittstellenbeschreibungssprache für anwendernahe Dienste in offenen Umgebungen vorgestellt. Anhand einer erweiterten Notation einer Service Interface Description Language (SIDL) soll so für den Benutzer von Diensten in offenen Umgebungen der Zugang und Aufruf dieser entfernten Dienste einheitlich ausgedrückt und strukturiert werden können. Dabei bietet die SIDL zudem auch Möglichkeiten, Dienstbeschreibungen etwa auch für die automatische Generierung von grafischen Benutzerschnittstellen zu verwenden.

[MeLa93b] M. Merz and W. Lamersdorf. Cooperation Support for an Open Service Market. In *Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing*. North–Holland, Elsevier Science Publishers B.V., December 1993.

Open communications technology allows the interconnecting of great multitudes of client applications with varieties of services in what can be considered a Common Open Service Market (COSM). In a COSM, application development can profit from existing services used as building blocks for the development of individual integrated applications. Decisive for the success of this software development process is the identification and relating of conforming cooperation partners with each other through an appropriate trading mechanism. The additional effort for a client to utilize remote servers or to switch between different providers of a distinct service is called transition effort. This effort should be reduced by the underlying support system as much as possible.
This paper seeks to derive design principles for distributed application development from an economics open market analogy. A Service Interface Description Language (SIDL) is presented as the basis for minimizing transition costs for distributed applications. It is used for the trading process as well as the creation of graphical local user interfaces for arbitrary remote services at binding time. Finally, the paper outlines the current status of a distributed prototype system which implements cooperation support for a COSM.

[MMM93] B. Mathiske, F. Matthes, and S. Müßig. The Tycoon System and Library Manual. DBIS Tycoon Report 212-93, Fachbereich Informatik, Universität Hamburg, Germany, December 1993.

This document provides a practical introduction to the interactive Tycoon system environment and its polymorphic libraries. It explains how to bind external C libraries to Tycoon programs and how to work with persistent stores. Moreover, it proposes formatting und naming guidelines for Tycoon programs.

[MaSc92] F. Matthes and J.W. Schmidt. Definition of the Tycoon Language - A Preliminary Report. Informatik Fachbericht FBI-HH-B-160/92, Fachbereich Informatik, Universität Hamburg, Germany, October 1992. (Also appeared as DBIS Tycoon Report 062-92).

This document defines the language TL that is used as the application and system programming language in the Tycoon database environment. The Tycoon project follows an add-on approach to generic database programming that emphasizes type-safe system scalability and extensibility.
TL is a polymorphic higher-order functional language with imperative features and inductively defined subtyping rules over types and type operator, extended by language constructs motivated by the needs of database programming.

[MMS92] F. Matthes, R. Müller, and J.W. Schmidt. Object Stores as Servers in Persistent Programming Environments - The P-Quest Experience. FIDE Technical Report FIDE/92/48, Fachbereich Informatik, Universität Hamburg, Germany, July 1992.

A promising approach to improve the quality of next-generation database programming languages is to strictly separate data modeling and data manipulation tasks from data storage issues. This separation greatly simplifies the development of tailor-made database languages and data models based on the generic services of a data model neutral object store protocol that provides efficient, transactional access primitives to persistent, shared and possibly distributed data objects.
As a first step towards this goal, this paper investigates the specific services which language clients require from their supporting object stores. It gives insight into the process of selecting a suitable object store for a polymorphic higher-order programming environment (P-Quest) and sketches the mapping of higher-level language objects like functions, environments and polymorphic aggregates, onto more primitive storage structures. The functionality of three candidate object stores (O2, Mneme and Napier POS) is evaluated based on the requirements imposed by the Quest language.

[MRSS92] F. Matthes, A. Rudloff, J.W. Schmidt, and K. Subieta. The Database Programming Language DBPL - User and System Manual. FIDE Technical Report FIDE/92/47, Fachbereich Informatik, Universität Hamburg, Germany, July 1992. (Also appeared as Informatik Fachbericht FBI-HH-B-159/92).

The database programming language DBPL is based on the notion of bulk type and iteration abstraction, supports data persistence and transaction procedures, and has Modula-2 as its algorithmic kernel.
This fist part of this document is intended for users of the DBPL system and gives an introduction into the DBPL language concepts and illustrates their use in a larger modular database application.
The second part gives some insight into the implementation of the DBPL system. As of today, the DBPL system exists in two fully source code compatible implementations, VAX/VMS DBPL and Sun DBPL. Both are written entierly in Modula-2 and consist of a compilation and a run time environment. The multi-user run time system is highly portable and runs under VAX/VMS 6.1, SunOS 4.1, IBM AIX 3.2 and IBM OS/2. Both DBPL systems can be integrated deeply into commercially available software development environments (NSE, SCCS, CMS) and provide optimized transactional multi-user access not only to type-complete DBPL databases but also to external relational databases via a fully transparent SQL gateway.

[MRSS92a] F. Matthes, A. Rudloff, J.W. Schmidt, and K. Subieta. A Gateway from a DBPL to Ingres: Modula-2, DBPL, SQL+C, Ingres. FIDE2 Technical Report FIDE/92/54, Fachbereich Informatik, Universität Hamburg, Germany, August 1992.

A gateway from DBPL (being a superset of Modula-2) to the commercial Ingres is described. DBPL extends Modula-2 in several ways, in particular, it introduces a new bulk and persitent data type "relation", and high-level relational expressions (queries) based on the predicate calculus. The gateway enables the user to write normal DBPL programs addressing Ingres databases. In contrast to typical implementations that embed SQL statements into a programming language, the interface becomes fully transparent to DBPL programmers: they need not to be familiar with SQL and Ingres. In this way the impedance mismatch problem is avoided. DBPL queries and other statements referring to Ingres tables are automatically converted into corresponding SQL statements, and the output from Ingres automatically becomes the property of the DBPL program. The gateway supports queries that refer both, Ingres and DBPL relations. This paper presents design assumptions of the gateway and implementation methods. In addition, we discuss design and implementation difficulties.

[NMM92] C. Niederee, S. Müßig, and F. Matthes. P-Quest User Manual. DBIS Tycoon Report 102-92, Fachbereich Informatik, Universität Hamburg, Germany, February 1992. (In German).

This document provides a smooth intorduction into the language concepts of P-Quest and the underlying persistence model.

[SchMa92] G. Schröder and F. Matthes. Using the Tycoon Compiler Toolkit. DBIS Tycoon Report 061-92, Fachbereich Informatik, Universität Hamburg, Germany, May 1992.

This document explains the structure and the use of the Tycoon compiler toolkit, a library of generic modules in the Tycoon system library implemented in P-Quest including a strongly-typed LL(1) parser generator.

[ScMa92] J.W. Schmidt and F. Matthes. The Database Programming Language DBPL - Rationale and Report. FIDE Technical Report FIDE/92/46, Fachbereich Informatik, Universität Hamburg, Germany, July 1992. (Also appeared as Informatik Fachbericht FBI-HH-B-158/92).

The database programming language DBPL is based on notion of bulk type and iteration abstraction, supports data persistence and transaction procedures, and has Modula-2 as its algorithmic kernel. This document describes the rationale behind DBPL and defines the elements of the language.

[STW92] K.-D. Schewe, B. Thalheim, and I. Wetzel. Foundations of Object-Oriented Database Concepts. Informatik Fachbericht FBI-HH-B-157/92, Fachbereich Informatik, Universität Hamburg, Germany, November 1992.

Mit der Entwicklung objektorientierter Datenbanken (OODBn) sollen die vielfach bemerkten Restriktionen des relationalen Datenmodells überwunden werden. Im Gegensatz zu diesem fehlt den Konzepten objektorientierter Datenbanken bisher allerdings sowohl eine zufriedenstellende formale Grundlage als auch ein allgemein akzeptiertes Datenmodell.
Ziel unserer gemeinsamer Forschungsarbeiten in Hamburg und Rostock ist die Entwicklung eines formal abgesicherten objektorientierten Datenmodells (OODM) als Beitrag zu einer einheitlichen mathematischen Theorie für OODBn.
Grundlegend für das OODM ist die Unterscheidung zwischen Werten und Objekten. Erstere werden durch Typen, letztere durch Klassen strukturiert. Ein zentrales Problem ist dabei die Bereitstellung eindeutiger Identifikationsmechanismen für Objekte. Wir lösen dieses Problem für Klassen, deren Objekte durch Werte vollständig repräsentiert werden können. Solche Klassen werden dann als werterepräsentierbar bezeichnet. Da eine Datenbank zu jedem Zeitpunkt nur endlich viele Objekte enthalten kann, ist Werterepräsentierbarkeit unter bestimmten Voraussetzungen entscheidbar, nämlich, wenn endlich repräsentierbare rekursive Typen existieren und alle benutzerdefinierten Integritätsbedingen sog. uniqueness constraints sind.
Ein Vorteil relationaler Datenbanken ist die Existenz eindeutig durch das Datenbankschema definierter generischer Veränderungsoperationen. Wir zeigen, daß diese Eigenschaft im OODM nur für werterepräsentierbare Klassen gilt. In diesem Fall aber wird sichergestellt, daß implizit durch das Schema gegebene Referenz- und Inklusionsintegritätsbedingungen automatisch bewahrt werden. Wir geben einen Algorithmus zur Erzeugung dieser generischen Operationen an und zeigen, daß dieser durch linguistische Reflexion implementierbar ist.
Dieses Ergebnis läßt sich auf ausgezeichnete Klassen statischer Integritätsbedingungen erweitern. Wir zeigen sogar, daß Integritätserzwingung im allgemeinen durch die Konstruktion größter konsistenter Spezialisierungen (GCS) möglich ist. Für eine beliebige Methode S und eine statische oder Transitionsintegritätsbedingung I kann gezeigt werden, daß die GCS S(I) immer existiert und bis auf semantische

Äquivalenz auch eindeutig ist. Mehr noch, GCSn sind mit Konjunktionen von Integritätsbedingungen, Vererbung und Verfeinerung kompatibel. Für benutzerdefinierte Transaktionen ist es allerdings nicht ausreichend, die hierin vorkommenden einfachen Veränderungsoperationen jeweils durch ihre GCSn zu ersetzen.

[JMSV91] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou. DAIDA: An Environment for Evolving Information Systems. Aachener Informatik-Berichte 91-18, RWTH Aachen, 1991.

We present a framework for the development of information systems based on the premise that the knowledge that influences the development process needs to somehow be captured, represented and managed if the development process is to be rationalized. Experiences with a prototype environment developed in ESPRIT project DAIDA demonstrate the approach. Specifically, the project has implemented an environment based on state-of-the-art languages for requirements modelling, design and implementation of information systems. In addition, the environment offers tools for aiding the mapping process from requirements to design and then to implementation, also for representing decisions reached during the development process. The development process itself is represented explicitly within the system, thus making the DAIDA development framework easier to comprehend, use and modify.

[Matt91] F. Matthes. P-Quest: Installation and User Manual. DBIS Tycoon Report 101-91, Fachbereich Informatik, Universität Hamburg, Germany, October 1991.

This document explains the installation and use of the P-Quest system (version 14) on Sun-4 hardware platforms.
P-Quest was developed at the University of Hamburg, Germany, and adds orthogonal persistence to the programming language Quest developed by Luca Cardelli at DEC SRC, Palo Alto, USA. The implementation of P-Quest was carried out in the ESPRIT-II basic research project FIDE and utilizes the Napier persistent object store provided by the University of St. Andrews, Scotland.

[MSS91] F. Matthes, G. Schröder, and J.W. Schmidt. VAX Modula-2 User's Guide; VAX DBPL User's Guide. DBPL Memo 121-91, Fachbereich Informatik, Universität Hamburg, Germany, December 1991.

This document describes how to compile, link and execute Modula-2 programs on a VAX under the VMS operating system. The VAX Modula-2 Compiler is based on an implementation of Modula-2 for the PDP-11 developed by a group under the direction of Prof. N. Wirth at the ETH Zürich.
This document also describes how to compile, link and execute DBPL programs on a VAX under the VMS operating system. DBPL is a relational database programming language that has Modula-2 as its algorithmic kernel. The DBPL Compiler is a fully upward compatible extension of the VAX Modula-2 Compiler.

[SSWB91] K.-D. Schewe, J.W. Schmidt, I. Wetzel, N. Bidoit, D. Castelli, and C. Meghini. Abstract Machines Revisited. FIDE Technical Report FIDE/91/11, Fachbereich Informatik, Universität Hamburg, Germany, March 1991.

The formalism of abstract machines has already been successfully used to establish a mapping from conceptual design modelled in the semantic data and transaction language TDL down to an implementation in the database programming language DBPL. We revise this formalism and compare it to predicative specifications as found in the formal specification language Z. The results show that the specification style of abstract machine operations, which is based on a slight extension of Dijkstra's substitution calculus, is in fact equivalent to the Z style. The main part of the paper addresses the problems of modularization, inheritance and refinement of abstract machines. We give formal definitions for the specialization and the refinement of substitutions and show their equivalence to the corresponding definitons in predicative specifications. We present some more partial results and conclude by mentioning some open problems.

[STWS91] K.-D. Schewe, B. Thalheim, I. Wetzel, and J.W. Schmidt. Extensible Safe Object-Oriented Design of Database Applications. Technical report, Universität Rostock, November 1991.

The formal foundation of object-oriented databases is still an open problem. In this paper a formalization of kernel object-oriented design concepts is achieved by an automatizable transformation into a specification language with a clear mathematical semantics. This approach allows object-oriented design to be extensible.
The preservation of advantages of relational databases requires the definability of generic operations for the insertion, deletion and update of single objects. It is shown that such generic operations can only be defined for value-representable classes. The possibility to detect value-representability contributes to the safety for object-oriented design.

[SWS91] K.-D. Schewe, I. Wetzel, and J.W. Schmidt. Towards a Structured Specification Language for Database Applications. FIDE Technical Report FIDE/93/59, Fachbereich Informatik, Universität Hamburg, Germany, July 1991.

Database application programs may be considered as good candidates for the application of formal specification methods, because much of the database and transaction semantics can be captured in terms of integrity constraints as well as pre- and post-conditions. Furthermore, the use of current transaction models as units of database state transitions favours the use of Dijkstra's substitutions.
Traditional DBMS solutions guarantee database integrity by expensive tests at transaction commit time. In the DAIDA project a different approach was taken interpreting database constraints as parts of formal specifications and by extracting correct database programs from such specifications. Database and transaction constraints are captured by an expressive semantic data model in the style of TAXIS, transformed into Abstract Machines and finally refined into AM versions that are equivalent to programs written in the strongly-typed database programming language DBPL.
The work reported in this paper evaluates the DAIDA experience and addresses the shortcomings of DAIDA. A new formal specification language SAMT is proposed that essentially allows to structure and modularize specifications and to capture particular classes of constraints by its type system. Moreover, SAMT is considered as a framework for both the specification of database applications as well as the refinement of specifications into executable programs, thus eliminating some of the complexity issues found in the multi-language approach of DAIDA.

[ScMa90b] J.W. Schmidt and F. Matthes. DBPL Language and System Manual. Esprit Project 892 MAP 2.3, Fachbereich Informatik, Universität Hamburg, Germany, April 1990.

The database programming language DBPL is a successor of Pascal/R and integrates a set- and predicate-oriented view of relational database modelling into the system programming language Modula-2. Based on integrated programming language and database technology it offers a uniform framework for the implementation and maintenance of database and application software.
This document describes the rationale for the design of DBPL, defines the elements of the DBPL language, and explains the use of the system software available for VAX/VMS systems. A small, self-contained application program is presented in chapter 5 to illustrate fundamental aspects of application programming in DBPL (data definition, data manipulation, modularization, user-interaction).
Chapter 3 gives a detailed description of the syntax and semantics of DBPL. It is

based on the Modula-2 Report by N. Wirth as published in N. Wirth: Programming in Modula-2, Springer-Verlag, Berlin, Heidelberg, New York, 3rd Edition, 1985.