Lecture Notes in Computer Science 760

Stefano Ceri Katsumi Tanaka
Shalom Tsur (Eds.)

Deductive and Object-Oriented Databases

Third International Conference, DOOD '93
Phoenix, Arizona, USA, December 1993
Proceedings

Springer-Verlag

Stefano Ceri Katsumi Tanaka
Shalom Tsur (Eds.)

Deductive and Object-Oriented Databases

Third International Conference, DOOD '93
Phoenix, Arizona, USA, December 6-8, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Stefano Ceri
Politecnico di Milano, Dipartimento di Elettronica
Piazza Leonardo da Vinci, 32, I-20133 Milano, Italy

Katsumi Tanaka
Kobe University, Department of Computer and Systems Engineering
Rokkodai, Nada, Kobe 657, Japan

Shalom Tsur
The University of Texas, System Ctr. for High Performance Computing
Balcones Research Center
10100 N. Burnett Road, Austin, TX 78758-4497, USA

CR Subject Classification (1991): H.2.3, D.1.5-6, I.2.4, F.4.1

ISBN 3-540-57530-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57530-8 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera-ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

The Third International Conference on Deductive and Object-Oriented Databases is a continuation of the two previous conferences in ~~this area. Its~~ ~~central tenet~~ is the continuing belief that the object-oriented and deductive paradigms for the modeling, organization, and processing of data complement each other rather than competing, and that the solution of problems involving massive volumes of complex data can best be attempted by utilizing the best of both approaches in an integrated fashion.

Central questions in this area are therefore: "How do we design a tool that presents the best of the object-oriented and declarative ideas, blended into one seamless form? How can the users of this tool express their problems in a combination of declarative and procedural features as their needs dictate it?" The search for answers to these issues forms a continuing quest and we have attempted to include papers in this volume that contribute towards this goal.

This volume contains twenty-nine papers. Three invited papers (David Maier, Kotagiri Ramamohanarao, Rainer Manthey) well represent the current efforts towards establishing the technology of deductive and object-oriented databases though concrete prototyping and product-oriented project experiences: Proxies, Aditi, and IDEA.

Twenty-six regular papers were selected out of a total of seventy submissions. Eleven of them were selected by the European Program Committee after a Program Committee meeting held in Milan on June 30; ten were selected by the American Program Committee; and five were selected by the Far East Program Committee. The outcome of the Far East and American Committee was solely based on the referee reports received for each paper and the discussions among the members of the committees. As a noteworthy feature it should be mentioned that the global evolution of electronic mail made it possible to conclude the discussions among the different chairs without ever having to physically gather at one place.

The editors wish to thank all of those who committed their time and efforts towards the success of this conference, either by submitting papers and/or by reviewing them.

December 1993

Stefano Ceri
Katsumi Tanaka
Shalom Tsur

Message of the General Conference Chairperson

It gives me great pleasure to welcome the Third International Conference on Deductive and Object-Oriented Databases (DOOD93) to the Valley of the Sun in Scottsdale, Arizona. This is the first time the DOOD conference has been held in the Americas. The notable success of the first two conferences in Kyoto, Japan, and Munich, Germany, established a challenging benchmark for us to meet. Judging from the quality of the papers that were accepted, I feel confident that this tradition of success is being continued. The original goal of the DOOD conference was to bring together researchers and practitioners who are dealing with two of the most promising areas of database research, deductive logic and object-orientation. This goal is still valid, especially as it appears that there will be industrial DOOD systems emerging in the near future.

The high quality of the papers presented at the conference is a direct result of the hard work and diligence of the three program committees and external reviewers under the supervision and guidance of the Program Chairs: Prof. Stefano Ceri, Prof. Katsumi Tanaka and Dr. Shalom Tsur. I would like to express my sincere thanks for their efforts in selecting twenty-six high quality papers and to the authors. I also want to express my thanks to the three invited speakers, Prof. David Maier, Prof. Kotagiri Ramamohanarao and Prof. Rainer Manthey. In recognition of the movement of DOOD out of the research laboratory and into industrial environments, the conference is also featuring two panels, one devoted to DOOD research directions and the other exploring potential application domains for DOOD.

An interesting innovation for this conference is that several papers have been nominated for inclusion in a special issue of the *Journal of Intelligent Information Systems*, dedicated to DOOD topics, that is to appear in 1994.

The conference also includes several special activities for spouses and attendees that are intended to allow participants to gain a deeper appreciation of the Arizona locale. This includes an all-day tour to the Grand Canyon, immediately following the conference.

An undertaking of this magnitude cannot succeed without the assistance of numerous dedicated people who give unselfishly of their time and efforts. I would especially like to thank the chair of the organizing committee, Dr. Forouzan Golshani, for his extensive contributions. Thanks also are due to Prof. Robert Meitz, the treasurer, Ms. Robin Fulford, chair of publications and publicity and to Mr. Ted Karren, chair of registration.

I would like to express my gratitude to the chair of the DOOD Steering Committee, Dr. Jean-Marie Nicolas, and to each of the Steering Committee members. Finally, I would like to thank Prof. Jack Minker, Steering Committee Chair Emeritus, who remains one of the most important driving forces behind the success of this conference.

December 1993 Oris Friesen

Program Committee Members

America

Anthony Bonner
(Univ. of Toronto)
Suzanne Dietrich
(Arizona State Univ.)
Sumit Ganguly
(Rutgers Univ.)
Narain Gehani
(ATT Bell Laboratories)
Michael Kifer
(SUNY Stony Brook)
Jean-Louis Lassez
(IBM TJWatson Res.Ctr.)
Jack Orenstein
(Object Design Inc.)
Raghu Ramakrishnan
(Univ. of Wisconsin)
Ken Ross
(Columbia Univ.)
Jehoshua Sagiv
(Hebrew Univ.)
Olivia Sheng
(Univ. of Arizona)
Oded Shmueli
(Technion, Haifa)
Ouri Wolfson
(Univ. of Illinios, Chicago)
Clement Yu
(Univ. of Illinios, Chicago)
Carlo Zaniolo
(UCLA)

Europe

Serge Abiteboul
(INRIA, Paris)
Peter Apers
(Univ. of Twente)
Elisa Bertino
(Univ. of Genova)
Francois Bry
(ECRC, Munich)
Georg Gottlob
(Tech. Univ. Wien)
Peter Gray
(Univ. of Aberdeen)
Klaus Dittrich
(Univ. of Zurich)
Giorgio Ghelli
(Univ. of Pisa)
Rainer Manthey
(Univ. of Bonn)
Jan Paredaens
(Univ. of Antwerp/UIA)
Joachim Schmidt
(Hamburg Univ.)
Marc Scholl
(Univ of Ulm)
Letizia Tanca
(Politecnico di Milano)
Patrick Valduriez
(INRIA, Paris)
Fernando Velez
(O2 Technology)
Laurent Vieille
(Bull SA)
Roberto Zicari
(J.-W. Goethe Univ.)

Far East

Qiming Chen
(Tsing-Hua Univ.)
Kazuhiko Kato
(Univ. of Tokyo)
Tok-Wang Ling
(Natl. Univ. of Singapore)
Hongjun Lu
(Natl. Univ. of Singapore)
Akifumi Makinouchi
(Kyushu Univ.)
Nobuyoshi Miyazaki
(Oki)
Shojiro Nishio
(Osaka Univ.)
Atsushi Ohori
(Oki)
Maria Orlowska
(The Univ. of Queensland)
Ron Sacks-Davis
(RMIT, Univ. Melbourne)
Toshihisa Takagi
(Univ. of Tokyo)
Kyu-Young Whang
(KAIST)
Kazumasa Yokota
(ICOT)
Masatoshi Yoshikawa
(Kyoto Sangyo Univ.)

Contents

Treating Programs as Objects: The Computational Proxy Experience

David Maier and Judith B. Cushing

Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
20000 N.W. Walker Road P.O. Box 91000
Portland, OR 97291-1000

Abstract. Migrating data to a new database model presents problems if there are existing application programs that must continue to access the data, bu that cannot be converted immediately. If the target database is object-oriented, such a legacy program can be encapsulated as an object or a message. We argue that some applications will benefit from further "reification" of execution instances as database objects. We introduce a "computational proxy" mechanism and our prototype implementation of it for computational chemistry codes. We conclude with a discussion of where declarative capabilities would have been a useful adjunct to object-oriented database features.

1 Introduction

Object-oriented databases provide type definition facilities that cope well with complex structures that arise in advanced applications, such as scientific computing. While transitioning datasets to an object-oriented environment can be non-trivial, providing for existing application programs to access the data in its new form can be a greater challenge. It is not always possible or feasible to convert those programs immediately to interface to the database directly, for several reasons:

1. Programming resources are not available to make the modifications.
2. There is not an appropriate application programming interface for the language in which the application is coded.
3. The source code of the program is not maintained or understood locally.
4. There are users of the program who still need to access existing datasets in the old format.

Thus, there is often a need to continue running an application program in its existing form, while providing a bridge to the data in its new form. We note that object-oriented databases are mostly introduced to supplant file-based data management, rather than replace an existing database management system. Hence, we direct our attention to existing applications with file-based data access.

One approach is simply to write a database application that accesses the appropriate data objects and creates an input file for the legacy program, then captures the output file and makes updates to the database. While this loose coupling of the program and the data may often be appropriate, for the domain of computational science where we are working, we desire a tighter binding. Runs of the program are tantamount to "experiments", and we want to record that the run took place, the results, and information about the run—rate of convergence, resource usage, exceptional conditions. Thus, we want the database to have ready access to information about program executions. Other application areas also exhibit this need to record program execution, such as the run of a design-rule checker against a circuit layout or a compilation in a CASE tool.

A more integrated approach—proposed more than once in the literature—is to leave the application program as is, and encapsulate it as a database object or a message of a database class. In the former case, the application is invoked when a message with the input data as an argument is sent to its wrapper object. In the latter, it runs when its corresponding message is sent to the object representing its input. Either approach assumes some way of calling out to the application or linking it in to the database executable, as well as constructing conversion routines between data objects and files. While "wrapping" a legacy application may work in some cases, we believe it will often be too limited an approach.

Capturing an external application program as a single object or message assumes a very input-output model, one where all the inputs are passed in as a unit, the program executes and the outputs are received as a unit. We believe this simple model can be inconvenient or inappropriate for many applications. We may want to separate supplying inputs from scheduling the execution of the program. In particular, when a computation is long, we might not want to wait while it executes. We would rather that the program run asynchronously from the database session, so we are free to do other things while it executes. Instead of treating the program execution as an atomic action, we may want to monitor its progress, in order to stop it, checkpoint it or observe intermediate results. Further, a single object or messages gives little help for organizing and gathering inputs or for structuring the translation process.

In this paper we offer a refinement of the naive wrapper approach that deals better with existing programs having complex inputs and long execution times. In addition to modeling the legacy program in the database, we also represent an individual invocation of the program as an object. Having such computation objects, or *Computational Proxies* as we term them, provides greater control over input assembly, scheduling invocation, execution monitoring and output processing. Furthermore, computational proxies offer a means to provide uniform interfaces to collections of programs whose inputs are semantically similar but syntactically diverse. The proxy mechanism is also a basis for constructing and managing suites and sequences of runs.

We are not alone in applying object-oriented databases to supporting scientific computing. A recent NSF report [CCT93] shows object-oriented approaches used for protein-structure data, medical research, macromolecules, global change data and scientific visualization. In particular, the MOOSE system [IL92] has dealt with modeling the complex inputs to a scientific simulation program.

Our main motivation was masking syntactic complexity, but proxies also serve to mask some details of processor heterogeneity and distribution, in the case where the underlying application runs in several dissimilar environments. From this vantage point, our work resembles other efforts in software systems to promote program interoperability, such as software "packaging" [CP91], middleware (see for example Bernstein [Be93]) and distributed object management systems [NWM93].

We report here on our use of computational proxies in the domain of computational chemistry, and then try to abstract lessons that can be applied in other domains, as well as indicating where more declarative formalisms would be a useful adjunct to object databases in support of our work.

2 Experience with Computational Proxies

The Computational Chemistry Database Project (CCDB), a joint effort of the Scientific Database Group at the Oregon Graduate Institute and the Molecular Science Research Center at Battelle's Pacific Northwest Laboratory, began as an exploration of the hypothesis that object-oriented databases could simplify the complex computing environment in which computational chemists find themselves. The resulting database infrastructure is predicated on providing not only relatively well understood data services but also extending object-oriented database functionality to provide computation services.

Ab initio molecular orbital methods apply quantum-mechanical techniques to molecular structure and energetics, solving the Schödinger equation to various levels of approximation. The solutions produce a wave function and associated electron density from which can be computed any observable molecular, property such as vibrational frequencies or electrostatic moments (dipoles, quadrupoles, etc.). Important inputs to these applications include an initial guess of molecular structure and a basis set of functions which together provide a starting point for iterating the Schrödinger equation. Other input parameters can be specified, depending on the particular application; these include the level of approximation to which to take the calculation, some maximum number of iterations, and the choice of a particular algorithm. The major outputs of the application include an optimized molecular structure, an energy value corresponding to that structure, and the corresponding wave function (also called electron density function or molecular orbitals). From the wave function are calculated the outputs of primary interest to non-theorists, for example, chemical properties such as electrostatic moments or hydrophobicity. In summary, these applications compute chemical properties directly from first principles using equations from quantum chemistry. Figure 1 gives an overview of inputs and outputs for computational chemistry codes.

A typical computing environment for a computational chemist consists of one or more of the application packages in fairly common use, such as Gaussian, GAMESS, MELDFX and HONDO. The application programs are very (100,000 to 300,00 lines of code), maintained remotely, computationally intensive, and

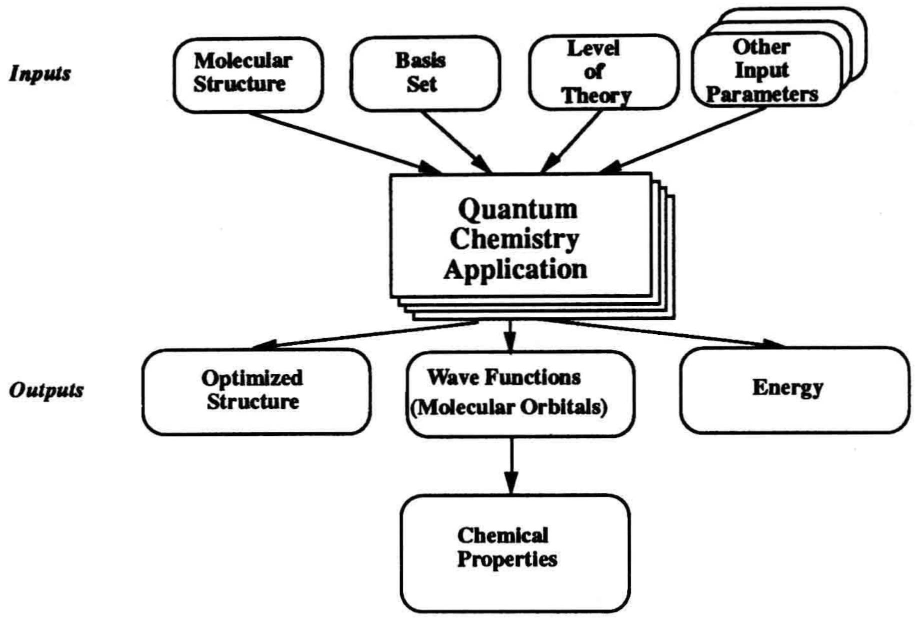

Fig. 1. Inputs and outputs for computational chemistry codes.

semantically similar but syntactically idiosyncratic. They require complex input files, create intermediate files of a gigabyte or more, and run on a variety of platforms from workstations to supercomputers. Output files are typically no larger than a few megabytes, but can be difficult to interpret. In the course of a single investigation, a chemist may generate hundreds of input and output files.

The considerable semantic complexity of the applications lies primarily in selecting input parameters appropriate for the subject molecule and desired results. A poor choice of input parameters will, at best, result in days or weeks of lost CPU hours. At worst, it can yield a plausible but incorrect result. The syntactic complexity lies in the relatively arcane formats of input and output files. Indeed, inputs or outputs of one application cannot easily be used as the input to another; considerable data transformation is required. Reformatting outputs of one program to use as inputs to another is a major problem for many chemists using these applications.

While much of the *semantics* of one application are transferable to another, the *syntax* is not. These applications were developed independently, over a long period of time; with few exceptions,[1] they are syntactically quite different. This *syntactic complexity* makes sharing of data between applications difficult. Figure 2 illustrates some of the conversions needed to run the same molecule through different application packages and compare outputs.

[1] GAMESS and HONDO are similar.

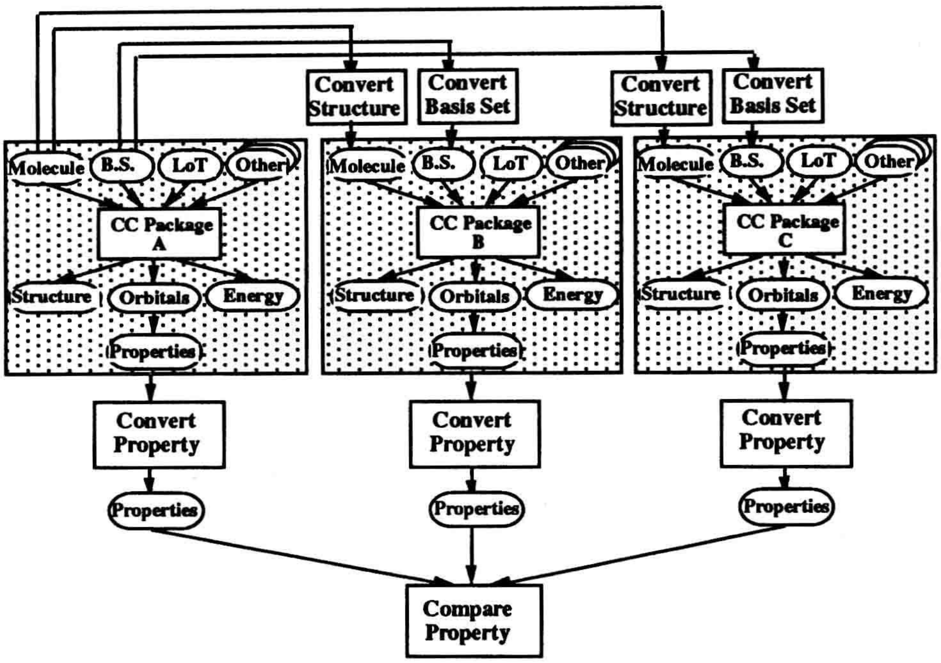

Fig. 2. Syntactical Complexity of Computational Chemistry Applications.

Our initial objective was to create a database of inputs and outputs of previous experiments to aid in selecting input parameters. In addition, the naming and managing hundreds of input and output files spread over multiple machines during the course of an investigation is too great a data management burden on the scientist. Continuing to use flat files is not a viable option for effectively sharing experimental results.

As a first step towards constructing the database, we designed a conceptual model covering the programs' major inputs and outputs. We chose and object-oriented database to implement that model, finding its language and modeling features (particularly encapsulation and direct representation of complex structures) to fit our requirements. An obvious additional advantage was that we could map conceptual-level classes, operations and hierarchies directly into counterparts in the database's data definition language (DDL) with little encoding. Figure 5 is a top-level view of the logical database design. This model was implemented in the ObjectStore object-oriented database management system [CMR+92b].

Experience using standalone programs to load the prototype database with experimental inputs and outputs, however, impelled us to seriously consider how to integrate the applications with the database and make the database the focal point for running computational experiments. Leaving the loading as a task separate separate from running an application would add to the chemist's overhead in running computational experiments and would inevitably prove error prone. We also wanted a more uniform interface to a diverse computational environment, and having the database separate from the applications did little to simplify the computing environment.

An obvious way to interface the applications to the database would have been to modify the programs to access and write to the database directly. The computational chemistry programs in question are simply too large, complex, and too often revised by others to make this approach feasible. A second alternative was to provide a "wrapper" for the computational applications, sending inputs to the wrapper and having it invoke application. However, this alternative is too simplistic; for example, it does not easily allow for automatically placing results of the computation back into the database, nor does it simplify monitoring of ongoing experiments.

Because the textual interface to these programs is easier to read and more stable than the application programs themselves, we decided to use the textual inputs and outputs as a basis for the interface, and to define an intermediate structure that would represent the computation in the database and help automate the control of ongoing experiments. We dubbed this mechanism a *computational proxy*. Computational proxies "stand-in", within the database, for computational experiments in preparation, currently in process, or recently completed.

Messages associated with the proxy class provide an interface to the computational programs used to run those experiments. The proxy encapsulates syntactic differences among semantically related applications, providing transformations of database items to application inputs and of application output into the database. Figure 3 shows this conceptual encapsulation of computational chemistry applications. When the user schedules a run, a proxy automatically transforms experimental attributes held in the database into textual inputs appropriate for a given application program. The inputs are shipped to the processor where the application is to run, and the application is invoked remotely on them. During the run, the user can interact with the proxy to halt and restart computation, view intermediate results (such as error terms) and monitor resource usages. Once the run has terminated, the proxy parses the outputs and places experimental results into the database in a form comparable to that of data for other applications. Thus, a database of experiment inputs and outputs is an immediate byproduct of launching experiments via the proxy mechanism.

Our first realization of the proxy mechanism consists of hand-coded programs that generated input files and parsed output files specifically for a given application. However, we feel that the user community should be able to register an application with the database descriptively by creating new objects, rather than by writing code specific to each application. We have thus focused our recent efforts on the ability to specify proxies declaratively. We call these specifications *templates*. The application descriptor, or *template*, defines the mapping between the domain-specific database and program-specific inputs and outputs.

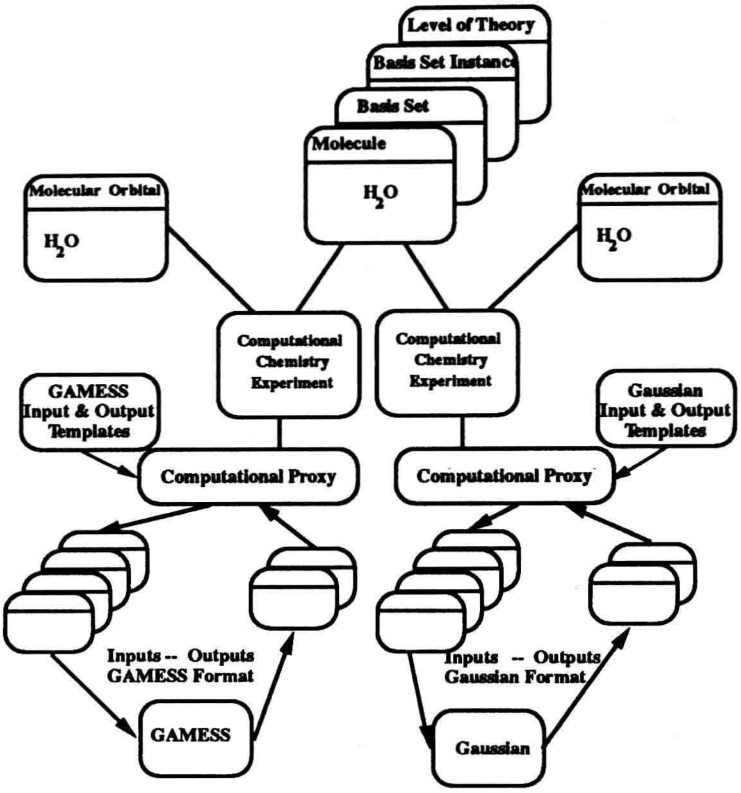
Level of Theory
Basis Set Instance
Basis Set
Molecule
H_2O
Molecular Orbital
H_2O
Molecular Orbital
H_2O
Computational Chemistry Experiment
Computational Chemistry Experiment
GAMESS Input & Output Templates
Gaussian Input & Output Templates
Computational Proxy
Computational Proxy
Inputs -- Outputs GAMESS Format
Inputs -- Outputs Gaussian Format
GAMESS
Gaussian

Fig. 3. Hiding syntactic complexity of computational codes.

We extended our logical data model to incorporate computational proxies. A computational proxy *represents* a process that is about to run, is running, or has run an experiment. A proxy *runs on* a *particular computer*, i.e., processor, that is connected to the same network services to which the proxy itself is connected. Any particular computer *is an instance of* some generic *computer platform*, e.g., the processor **coho** is a **Sun4** computer platform. If a proxy object *runs on* a particular processor, we say that the *experiment represented by that proxy runs on that processor*. In order for an experiment to run on a processor of a particular type of computer platform, the application it *uses* must be both *available for* the corresponding platform and *installed on* the particular processor. Figure 4 illustrates these conceptual relationships.

To date, we have implemented a computational proxy mechanism that interfaces directly to the GAMESS package. This prototype currently runs under C++ and ObjectStore on distributed Sun SparcStation2 platforms. We are currently working on the database structures that will allow the non-programmatic specifications of proxies for new applications [C93].