The Gist of GIUKU
Graphical Interactive Intelligent Utilities for Knowledgeable Users of Data Base Systems

Michel Kuntz
ECRC, Arabellastraße 17, 8000 München 81, Germany
email: kuntz@ecrc.de

Abstract

Synoptic description of GIUKU: its rationale, its main functionalities, its novel features, a comparison to related work, and a discussion of its current status — a fully implemented prototype available for use.

1 Introduction

GIUKU is a contribution to the research efforts aimed at improving the tools that facilitate communication with, and operation of, data base systems (DBSs). More precisely, it addresses the needs of knowledgeable users and thus supports the exploratory programming involved in developing new kinds of scientific and technical applications now being tackled with state-of-the-art DBSs, like MegaLog [4] and EKS–v1[22], the two systems for which GIUKU's current implementation is specifically tailored. MegaLog is a persistent logic programming system for developing large-scale data and knowledge base management systems. EKS–v1 is a knowledge base management system that provides extensive and efficient inferential capabilities on large amounts of data: deduction rules, integrity maintenance, pre/post-conditional updates, and hypothetical reasoning. EKS–v1 is implemented in MegaLog.

In [21], Tsur mentions important classes of such non-traditional applications and describes the context in which tools like GIUKU are urgently required. Such tools are significantly different both from those that are intended for "naive end-users" and from traditional 4GL–style application frameworks. They must offer assistance for fast prototyping (designing, writing, testing, and modifying) of schema definitions, deduction rules, integrity constraints, declarative queries, and procedural code; application independence without isolation from applications; and the ability to scale up to realistically large data bases. Such tools naturally have many of the characteristics of a programming environment. Since they rely critically on graphical direct-manipulation interfacing capabilities, they also have many of the features of a graphical user interface. Such tools leverage the knowledge they possess of the back–end DBS and the applications under development in order to provide *intelligent* assistance.

This article is an introductory overview of GIUKU. The next section sketches the rationale behind GIUKU. Section 3 summarizes the GIUKU facilities, tool by tool, as seen from the user's perspective, and highlights novel features. A comparison to related work in Section 4 positions GIUKU with respect to other research pursuing similar goals. Finally, a brief discussion of GIUKU's current status and future plans is the topic of Section 5.

2 Rationale

GIUKU's longer-term goal is the development of intelligent graphical interaction techniques for DBSs. However, interactive software is more like a pudding than a theorem — the proof of its validity is based on the extent of its acceptance by users. Validation by implementation is thus necessary but not sufficient: the good qualities of the implemented software must be further acknowledged by the users for whom it was intended. From this

empiricist view of building interactive software, several informal principles can be derived, which have guided the design of GIUKU.

Principle 1: do not create solutions in search of problems. This pitfall can be avoided by focussing on actual user needs and deriving requirements from an analysis of *their* tasks (cf. [14]): which are the most difficult to carry out and which have to be accomplished most frequently ?

Principle 2: target users need to be "knowledgeable". GIUKU is **not** for "naive" users. This negative statement needs to be made explicitly because many people still strongly associate direct-manipulation graphics (DMGr) with such users, believing that DMGr makes a DBS easy to use for them but is superfluous for anyone else. Both beliefs are wrong. DMGr does allow users to interact with a system without the need to concentrate on recalling their knowledge about language syntax – which is by no means negligible. It does not, however, overcome a lack of knowledge about the semantics of the following aspects of the system: the application domain in question, the data model and associated languages of the DBS, the general interaction paradigm involved, and the particular interactive tools used. "Knowledgeable" users are thus people who have at least some knowledge in all four areas just mentioned. "Naive" users are those who have no knowledge in one or more of these areas.

Principle 3: DMGr is good, *intelligent* DMGr is better. A user interface is a mediator between two models of the interactive system. The first model is the one that corresponds to how the system actually is – in the optimal case this reflects what the designers and implementors had in mind and what the documentation describes. The other model is the user's mental model of the system – how the user perceives the system, what the user knows about it. An "intelligent" user interface is one that has and uses knowledge about the interactive system to enhance its own ability to carry out the mediating job, thereby relieving the user of part of this responsibility.

Principle 4: DMGr is complementary to textual languages. DMGr is to be used with, and not as a replacement for, textual languages. Users can employ any combination of DMGr and textual-language operations that they desire at any time and for any task. The interactive tools are able to translate back and forth between the two representations.

Principle 5: no tool is universal. No single interfacing tool can be optimally adapted to all tasks and all kinds of data. Therefore, a collection of diverse utilities is required, from which the right one can be chosen each time, but which all fit together to form a coherent environment.

Principle 6: tools should be independent of the application domain. Similarly to its back-end DBSs, GIUKU is independent of any particular application domain. This clearly also makes it independent of any individual data base in that domain, as well as any single given application for such a data base.

Principle 7: stick to already available interactive technologies. Many features of interactive tools are constrained by the technology available for implementing them. In order to enable broad availability, it was decided in the case of GIUKU to forego all attempts at using technologies that will remain non-standard on garden-variety workstations in the immediate future (e.g., vision-controlling helmets à la virtual reality, data gloves, eye-movement trackers, fully 3-D displays, speech generators).

3 Basic Description

GIUKU's functionalities are available through 14 different tools, most of which can be used independently of the others. However, many tools do in fact complement each other — several invocations of the same utility or of different ones can be operated simultaneously, directly supporting the user's need to work on multiple tasks at the same time. GIUKU can be seen as integrating three kinds of subsystems: (a) those that support direct manipulation of the data base at both the schema and data levels, (b) those that support programming in an extended sense: expressing *all* the language constructs that contribute to defining a deductive data base and its applications (schema definitions, deduction rules, integrity constraints, declarative queries, and procedural program code), and (c) those that make the interaction of the user with GIUKU more effective. However, the distinction between functionalities of type (a) and (b) is somewhat artificial for two reasons: many GIUKU

tools can produce the textual constructs of (b) from the graphical representations of (a) and vice versa, and the two kinds of functionality are inter–invocable: GIUKU tools can be called from within textual "code" and textually defined processing (notably, application–specific program code) can be automatically invoked as part of the interactive GIUKU utilities. Thus, the following descriptions are grouped according to schema–level tasks, data–level tasks, and tasks concerned with using GIUKU itself. Space limitations make it impossible to go into any degree of detail and to include screen snapshots illustrating the look and feel of most of these tools.

3.1 schema–level tasks

1: gdrs — defining new relation schemas
gdrs incorporates knowledge about well-formedness of relation schemas and so automatically leads to correct definitions. It can also generate the textual version of the definition. Its analysis capability compares the new definition to the rest of the data base schema in order to catch potential conflicts, redundancies, resemblances, etc. **gdrs** can also generate "dummy" data, i.e., automatically populate the new relation with syntactically correct tuples, making it easy to test application code without the drudgery of manually entering test data. Figure 1 shows an invocation of **gdrs** at a point in time where the user had not yet decided on the new relation's

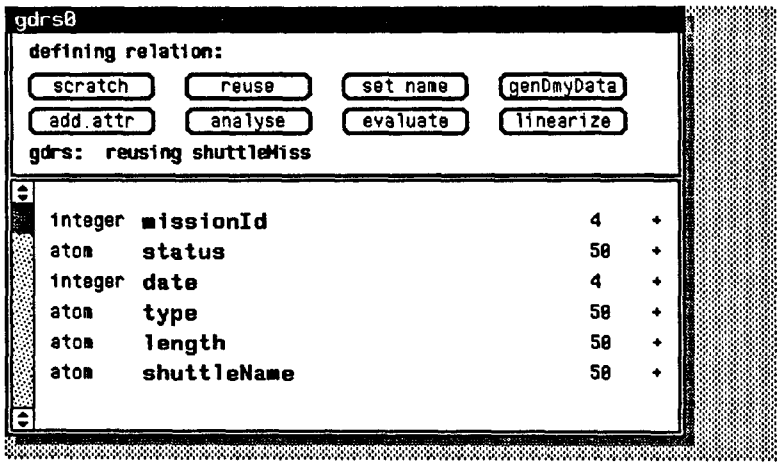

Figure 1: The GIUKU utility for defining new relations

name, but had already determined that the most effective way to define it was to edit the definition of a previously existing one, "shuttleMiss". The lower pane displays this definition in its graphical representation. Each line defines one attribute: its type, name, width, and key status. All items are mouse-sensitive. Clicking on them pops up a menu of possible operations for the *name* item or possible values for the other three items.

2: gcat — manipulating the catalogue
GIUKU's **gcat** facility displays a three-column table with relation names, arities, and cardinalities. Relation names are mouse-sensitive, and the associated pop-up menu provides numerous functionalities. Some menu items are a convenient operand-first way to invoke certain tools available elsewhere in GIUKU. The *attributes* item creates a window containing a tabular display of the attribute information. Other items help users write correct application code on the first try: they invoke template generators for tuple-at-a-time manipulation and relational algebra.

3: gsu — graphical syntax utility
gsu provides a graphical representation, based on abstract syntax trees and complete with powerful direct-manipulation editing capabilities, for the textual-language constructs used in building data base applications: deductive rules, integrity constraints, declarative queries, and procedural code. **gsu** shifts some of the cognitive complexity involved in using such languages when writing the initial version of a program from the programmers themselves to intelligent interactive tools at their service. The graphical representation in the **gsu** workspace can be imported from another GIUKU facility, or loaded into the workspace from one of the **gsu** libraries. It can

also be any textual expression already existing in the DBS i/o window. Besides extensive editing capabilities, such a graphical representation can be evaluated, linearized, and checked for well-formedness.

4: gbUDPs — cross referencing 1
gbUDPs displays an alphabetically ordered list of all the predicates defined in the files of source code that have been compiled during the current session. Predicates can be clicked on, to display the name of the file that the predicate is defined in and a tree showing all the predicates that call it, the predicates that call those predicates, and so on, until either a recursive call or a top-level predicate is reached.

5: gbDBManips — cross referencing 2
gbDBManips supplies information about how and where the data base is manipulated in application code. This is done by analyzing the source code and storing the results in a relation which users can then browse and/or query at their convenience.

3.2 data–level tasks

6: gbSummary — browsing 1
Summarizing is browsing a summary of the relation in question. Intuitively, a summary of a collection of tuples (i.e., of a relation) is a succinct representation of those tuples that highlights any similarities they may have and thereby organizes the collection in a meaningful way. The overall summary browsing process consists of two interleaved phases: construction of the summary tree from the raw data in the collection, and display and interactive manipulation of the tree. Summarizing combines features from summaries as known in the area of statistical DBSs [15], the NF^2 *nest* operator [18], and Tsur's concept of data dredging [21]. Figure 2 shows one possible state of the summary of the data concerning a wine cellar, taken from Fig 1.1 on page 4 of Date [6] less the last column there, "comments". Each node (except the root) in the summary tree has a complex label.

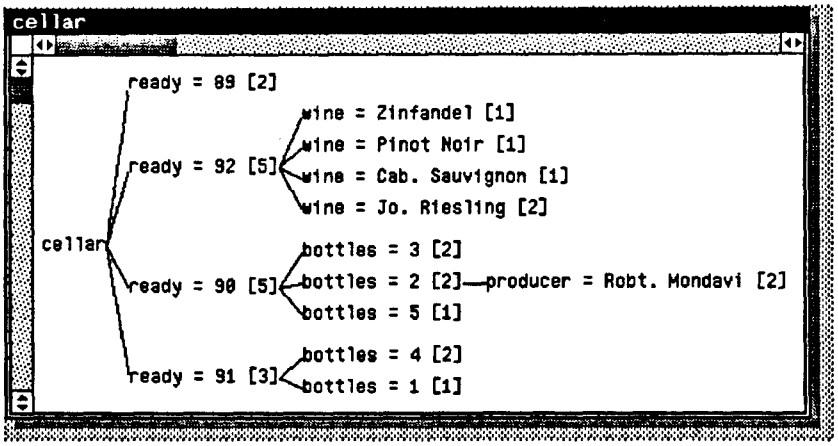

Figure 2: One possible summary tree for wine cellar data from Date

It shows the summarizing attribute itself, the value of it that was used in the selection operation that produced the sub-relation represented by the given node, and (in square brackets) the cardinality of that sub-relation. In Figure 2, the "ready" attribute was used as the first summarizer for the cellar relation, either because this is what the user manually asked for or because it was automatically found to be best.

7: gbTable — browsing 2
gbTable presents data in a tabular format, but gives users the added ability to manipulate them, since very often when users browse data, their overall goals in interacting with the DBS are not exclusively passive — they usually intend to single out some of the data from the rest for later reference and/or for further phases of processing. The mark-and-buffer capability directly supports this selection activity. Like the other GIUKU browsers, **gbTable** also supports a "hook" on which users can hang application-specific code.

8: gbTree — browsing 3
Tree browsing is a graphical facility specially adapted for visualizing data structures whose values in fact constitute a tree as far as the application semantics is concerned, even though they are stored in a relation. The values contained in two attributes can be interpreted as parent and child pairs. **gbTree** provides an ordered, scrollable list of all the parent node values, as well as a window in which the graphical tree structures are displayed. When users click on any parent node value in the list, the tree rooted at that value is shown in the graphical window.

9: gbectr — handling NF^2 relations
gbectr ("graphical browser-editor for complex-term relations") is a graphical interaction tool for manipulating relations whose attribute values may be complex terms (lists, literals, clauses). The tuple tree graphical representation used to display such data both portrays the connectivity of components more clearly and makes more efficient use of screen space. The browsing capability includes: sequencing through the relation, selective expansion of sub-terms, and a copy-and-buffer functionality. The querying capability exploits the expressive power of logical variables and unification, while extending the *retrieval-by-reformulation* approach [23, 9] to operations on complex data. Instead of being the graphical representation of an actually existing tuple, the data structure displayed in **gbectr** can define a "filter" that is used to select from the operand relation only those tuples that unify with it. Updating is also supported.

10: gqfl — simple graphical querying
In **gqfl**, a query's graphical representation is a dynamically modifiable collection of "atomic conditions", each of which involves four elements: the type, the attribute itself, a comparison operator, and a final field used for projection or selection. Atomic conditions can be inserted, deleted, and cloned. Users can repeatedly modify the collection of atomic conditions and then invoke evaluation and browsing of results. Queries can be saved in a library or linearized (transformed into text) for incorporation into application code.

11: gupdate — insertion and deletion
Each row in the **gupdate** workspace concerns a single attribute of the relation being updated and includes: a compact reminder of its definition, the attribute's name and either a dotted line or an already filled in value. If values are typed in by the user, **gupdate** immediately verifies that they conform to the attribute's type and width, and erases incorrect ones on the spot. Values can also be selected from pop-up menus.

3.3 using GIUKU itself

12: giuku — top–level interaction windows
GIUKU has two windows that are always available: the anchor window consists of an ideogram icon from which the GIUKU main menu pops up and a global message area; the GIUKU bulk displayer window is used to display relatively large quantities of information (e.g., a textual–language construct generated automatically from a graphical representation) without cluttering up the window dedicated to normal textual i/o with the back–end DBS.

13: ginfo — on-line help and documentation
ginfo uses a tree of topic nodes, which mirrors the structure of the GIUKU tools themselves. Each node can be sprouted in order to display its children and thus move down to the next level of detail. One eventually gets down to nodes representing individual buttons, labels, menus, etc. On any node at any level, users can invoke the "explain" functionality, opening a text window containing arbitrary amounts of information about the topic concerned.

14: gCP — global control panel
gCP is the GIUKU Control Panel. This is a facility similar to the one with the same name available on the Macintosh or also to the SunView Defaults Editor. It enables users of GIUKU to customize it, making it suit their own particular needs or tastes better. Examples include such considerations as: which font should be used in **gbTable**, whether a slow but screen-space-optimal or a fast but screen-space-wasteful algorithm should be used for laying out the contents of **gbTable** windows, and how many children are sprouted from a tree node without first asking the user to choose a subset from all the possibilities, which is applicable to several tools.

4 Related Work

GIUKU lies in the intersection of work on data bases, software engineering, and graphical human-computer interaction. Graphical user interfaces to DBSs have been around for some time (see the references given in [10, 11] for pointers into the literature). Programming environments with graphics ([16] provides a summary of this area as well as references) are nothing really new either. A graphics-based environment specifically geared to support **data base** programming *is* however a novel development. Efforts already reported in this area are few in number: OOPE [5] and the GemStone Visual Tools [2]. "Knowledge editors" such as [1, 17, 20, 7] make good use of graphics but do not focus on programming and are not part of a DBS as understood here. [12] mentions the idea of a programming environment for deductive DBs but does not develop it beyond a list of desirable features. The PICASSO Application Framework [8] is more analogous to the PCE toolkit (see Section 5) than to GIUKU itself. The IRIS graphical browsing and querying facilities [13] provide a single, unified DBS interface (rather than a collection of diverse utilities like GIUKU) and are not designed to support programming as GIUKU is. The work on DBS interfaces in the KIWIS ESPRIT Project [19] aims to cater to end-users who need not be knowledgeable.

How does GIUKU compare with OOPE and the GemStone Visual Tools ? Only the GemStone Visual Schema Designer has been described in the literature. It allows users to define new classes and browse existing ones. Its functionality is in this respect roughly equivalent (*modulo* the difference in data models) to GIUKU's **gdrs** and **gcat** tools. The OOPE browser gathers together functionalities that correspond partially to GIUKU's **gdrs**, **gcat**, **gbTable** and **gupdate** tools. The OOPE journal is a versioning facility for which there is no equivalent in GIUKU. The OOPE applications manager and queries manager seem to provide services roughly similar to the libraries associated with GIUKU's **gqfl** and **gsu**, but are not otherwise comparable to the latter two subsystems. The GIUKU tools **gbSummary**, **gbTree**, **gbectr**, **gbUDPs**, **gbDBManips**, **ginfo**, and **gCP** apparently have no counterparts in OOPE.

5 Discussion

GIUKU's current status is that an up-and-running implementation exists that is the result of a first loop through the fast-prototyping cycle. All the functionalities summarized above have already been implemented, using MegaLog itself and a version of the PCE graphics toolkit [3] which sits on top of either SunView or XView. The current implementation (about 15,000 lines of MegaLog) is for the most part fast enough for comfortable interaction even on aging Sun-3s. It has been used mainly with MegaLog. Certain parts of it still require modifications and/or extensions for best use with EKS–v1. To date it has been tested successfully on a dozen small data bases (largest relation: 50,000 tuples) constructed independently with no kludges to make them suitable for GIUKU. It has been undergoing alpha testing at ECRC and has been transfered to one of ECRC's shareholder companies for evaluation. The GIUKU binaries are available for research and educational purposes.

Work is continuing to further enhance the system. Aside from incorporating modifications and enhancements arising from user feedback, and besides numerous local extensions to the functionality of the already existing facilities, plans for the future of GIUKU concern four main areas: completing the EKS–v1 version of the system, adapting to operation in a multi-user context, un-hard-wiring as much as possible of GIUKU's own knowledge, and extending the current version to include more intelligent support for programming and deduction. Such enhancements are necessary in order to reach GIUKU's goal of real–world validation through actual use and acceptance by knowledgeable users of MegaLog and EKS–v1. Beyond the above enhancements, two complementary directions should be explored: relaxing design principle 7 in Section 2 to exploit exotic interaction technologies, and determining how the combination of deductive and multi-media capabilities in the same DBS influences the user interface(s) that such a system should have.

References

[1] Glenn Abrett and Mark Burstein. The kreme knowledge editing environment. *Int. J. Man-Machine Studies*,

27:103–126, 1987.

[2] Jay Almarode and T. L. Anderson. Gemstone visual schema designer. Technical report, Servio Corp., 1990.

[3] Anjo Anjewierden. Pce-prolog 1.0 beta, reference manual. Technical Report ESPRIT Project 1098, University of Amsterdam, 1986.

[4] Jorge B. Bocca. Megalog - a platform for developing knowledge base management systems. In *Proc. 2nd DASFAA*, Tokyo, April 1991.

[5] Patrick Borras, Anne Doucet, Patrick Pfeffer, and Didier Tallot. Oope, the o2 programming environment. In *Proc. 6. Jour. Bases de Donnees Avancees*, Montpellier, Sept 1990.

[6] C. J. Date. *An Introduction to Database Systems, volume I.* Addison-Wesley, Reading, MA, 1990.

[7] Marc Eisenstadt, John Domingue, Tim Rajan, and Enrico Motta. Visual knowledge engineering. *IEEE Trans. on Software Engineering*, 16(10):1164–1177, October 1990.

[8] Lawrence Rowe et al. The picasso application framework. Technical Report UCB/ERL M90-18, UC Berkeley, Electronics Research Lab, 1990.

[9] Gerhard Fischer and Helga Nieper-Lemke. Helgon: Extending the retrieval by reformulation paradigm. In *Proc. CHI 89*, pages 357–362. ACM SIGCHI, May 1989.

[10] Michel Kuntz and Rainer Melchert. Ergonomic schema design and browsing with more semantics in the pasta-3 end user interface for e-r dbmss. In *8th E-R Conf.*, pages 263–278, October 1989.

[11] Michel Kuntz and Rainer Melchert. Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power. In *Proc. VLDB 89*, pages 97–105, 1989.

[12] T. K. Lakshmanan. Towards a logic programming environment for database systems. In *Proc. NACLP'89 Workshop on Logic Programming Environments.*, pages 78–80, Cleveland, October 1989.

[13] Ashok Malhotra, Luanne Burns, Gary Sockut, and Kyu-Young Whang. Iris: Interactive repository interface services. Technical Report RC 16943, IBM T. J. Watson Research Center, 1991.

[14] Donald Norman and Stephen Draper (eds.). *User Centered System Design.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[15] G. Ozsoyoglu, V. Matos, and Z. M. Ozsoyoglu. Query processing techniques in the summary-table-by-example database query language. *ACM Transactions on Database Systems*, 14(4):526–573, 1989.

[16] Steven P. Reiss. Interacting with the field environment. *Software Practice and Experience*, 20(6):89–115, June 1990.

[17] W.-F. Riekert. The zoo metasystem: A direct-manipulation interface to object-oriented knowledge bases. In *Proc. ECOOP'87*, pages 145–153. AFCET, June 1987.

[18] Mark Roth, Henry Korth, and Abraham Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[19] F. Staes, L. Tarantino, B. Verdonk, and D. Vermeir. Supporting user interactions with oodb's: A declarative approach. In *DEXA 91*, pages 210–215, August 1991.

[20] Loren Terveen and David Wroblewski. A collaborative interface for editing large knowledge bases. In *Proc. 8th Nat'l Conf. on Artificial Intelligence*, pages 491–496. AAAI, 1990.

[21] Shalom Tsur. Deductive databases in action. In *SIGMOD 91*, pages 142–153, May 1991.

[22] L. Vieille, P. Bayer, V. Kuechenhoff, and A. Lefebvre. Eks-v1, a short overview. In *Proc. AAAI-90 Workshop on Knowledge Base Management Systems*, Boston, July 1990. AAAI.

[23] M. D. Williams. What makes rabbit run ? *Int. Journal of Man-Machine Studies*, 21:333–352, 1984.