Reprinted from:

AAMSI CONGRESS 83

Proceedings of the Congress
on Medical Informatics

Supporting Sponsor:
Kaiser Foundation Hospitals

Edited by:

DONALD A.B. LINDBERG, M.D., Sc.D.
Information Science Group
University of Missouri
Columbia, Missouri

EDMUND E. VAN BRUNT, M.D.
Kaiser-Permanente Medical Care Program
Oakland, California

MICHAEL A. JENKIN, M.D.
Computer Sciences Corporation
Herndon, Virginia

Volume 1
ISSN 0737-4194

American Association for Medical Systems and Informatics, Publishers
Suite 402, 4405 East-West Highway, Bethesda, MD 20814 301/657-4142

Printed in the United States of America

ENCOUNTER DATA SYSTEM (EDS) - AN IIS CASE STUDY

Daniel S. Keller
Anthony I. Wasserman

Section on Medical Information Science
University of California
San Francisco, CA 94143

A software system has been developed for a dermatology clinic's administrative data management. The system performs data entry and retrieval for generation of reports regarding patient/physician encounters, diagnoses made, and clinical procedures conducted. The virtues of the system stem from the "componentry" approach to software development, and from the high quality of the components used. Such systems can thus be constructed quickly and operated at low cost and are very flexible in the types of functions which they can perform.

Interactive Information Systems (IIS)

Interactive information systems are an important class of computer applications [1]. In an IIS, users enter data, typically via CRT terminals. Later these data can be recalled for editing, or for organization into reports and displays. Users are thus provided with conversational access to databases. Such systems can be described as having three components: a user/program dialogue, a database, and operations mapping the dialogue onto the database [2].

Because IISs are so common, we have sought to develop software components that facilitate their design and construction. This paper describes the component approach and its benefits. Space limitations prohibit either a lengthy exposition of the tools or the application systems built with them.

Componentry Software Methodology

Software componentry is an approach to systems development which makes use of an inventory of standardized parts. Programmers need not rewrite portions of code which perform similar functions, but may instead construct systems from tested, general-purpose program components. In contrast to "programming-in-the-small" (conventional programming) this approach has been termed "programming-in-the-large" [3]. The primary advantage is reduced development and validation time. Another advantage is the "decision-hiding" [4] that can be achieved; implementors are relieved of many detailed decisions such as data structure design. A disadvantage is that such generalized elements are not optimized for each specific use to which they are put.

EDS - An IIS Case Study

The Encounter Data System (EDS) is undergoing installation at UCSF's Dermatology Clinic as an administrative tool. It is a specialized software system constructed from an inventory of standard program parts. This approach is consistent with the philosophy of UNIX™ [5], the operating system on which EDS runs. UNIX provides a

---

Unix is a trademark of Bell Laboratories.

number of services to EDS and hence might be considered one of the components. These services include the "pipe" mechanism for interprocess communication, and the "termcap" terminal capability database which enables EDS to operate independently of the hardware characteristics of any specific terminal.

The other components of EDS are "editform", "Ingres", and "driver". Editform is a set of routines which handle the screen/keyboard user interaction. It provides messages, prompts, and data verification. Editform control files (termed "formfiles") can be prepared in minutes, allowing configuration for different applications. A form is an electronic embodiment of a paper form. It is a formalized questionnaire containing blanks ("fields") to be filled in. Once a form has been filled in, the data are sent to the database to be stored. Later, they may be recalled and modified.

Table 1 shows a sample formfile. Formfiles describe data entry screens and associated database actions. They consist of lines of various kinds. Each line is an ordered list of colon-delimited descriptive items. Formfile lines are described in Table 2.

```
# Form for entering a new encounter.
#
APPEND:encounter
GET:unitno
GET:visitdate
H:0,0:Unit number <unitno>|^Enter new encounter.|Visit date <visitdate>
::0,0,6:I:encounter:unitno:unitno:123456:LFK:LFK::
::0,0,15:I:encounter:visitdate:visitdate::D:D::
:Encounter type:4,30,4::encounter:enctype::699:IV:IV:100:699-902
100:verify encounter type code:Error -- unknown encounter type code.
:Physician code:6,30,4::encounter:mdno:mdno:P666:TV:TV:110:
110:verify physician on-file:Error -- unknown physician code.
:Next visit date:8,30,15::encounter:nextvisdat:::D:D::
:Special case code letter:10,30,1::encounter:speccase::Z:TF:TF::
T:10,20:Comments or other descriptive text.
::11,0,48::encounter:text::Prescribed drug X.:TV:TV::
```

Table 1 -- A typical formfile.

Ingres [6] is a relational database management system which handles the storage and retrieval of data received from editform. Ingres runs as the "backend" of an interactive information system, where editform collects the input at the "frontend". Ingres provides for the definition of permanent and temporary relations, for relational calculus-like operations of selection, projection, and join, and for assignments to individual attributes within a relation.

"Driver" is a program that permits editform to talk to Ingres. Different versions of driver have been written which interface to to other databases and to ordinary files. The exact configuration of components for any particular installation varies according to availability at each host site. In a typical version, driver and editform run as a single process and communicate with Ingres via the Unix pipe mechanism.

The advantage of the component approach is now apparent. There is no need to perform extensive physical database design; all that is needed is the logical database design expressed as a set of normalized relations. There is no need to program all of the user/program input/output operations; these are simply defined as editform inputs and outputs.

Because editform and Ingres have been thoroughly tested and extensively used, systems built with these tools have far fewer errors than those built in a traditional manner. Development time is significantly reduced, since much of the system specification can be captured with little coding effort.

starts with	action	description
#	comment	Editform ignores these lines.
H	header	Literal and variable text to appear in the header area of the screen. The "|" characters delimit an item to be centered. The "^" character causes the item to be displayed in reverse video.
T	text	Literal and variable text to appear in the data-entry area of the screen.
:	field def	Defines a data-entry field. These lines consist of 11 items. Only the second is not optional. These items are: displayed label; coordinates and width; whether invisible, required, or read-only; the relation in the database to which the datum belongs; the name of the attribute within the relation; the name of the variable within the form which contains the datum; an example in case the user asks for help; the display format (listed in Table 3); the storage format; verification routine number(s); and numerical validity range.
PRE	form preproc	Actions to perform prior to editing fields, e.g. database retrievals, variable initialization.
POST	form postproc	Actions to perform when the user is done entering/editing.
integer	proc def	Defines a form preproc, form postproc, or field verification routine. A string is provided for display in the message area of the screen in case the routine fails.
GET	fetch ifc	Tells Editform to fetch from the interform communication (ifc) area a variable which was stored there by a SAVE in a previous form.
SAVE	store ifc	Tells editform to store into the interform communication area (ifc) some variable for use by a later form.
APPEND	d.b.append	Appends a record from the edited form into the named relation in the database.
UPDATE	d.b.replace	Replaces a record from the edited form into the named relation in the database.

Table 2 -- The kinds of lines which can appear in formfiles.

In this implementation, the data stored are of eight types: patients, encounters, diagnoses found, clinical procedures conducted, doctors, a table of diagnosis codes, a table of procedure codes, and a table of special case codes. A given patient may have several encounters, each with perhaps a different physician, different diagnoses, different clinical procedures, and so forth. Also, an encounter may be flagged with a special case code, for example if it were of interest to a particular research project.

Diagnoses are entered using the SNODERM [7] encoding scheme. Clinical procedures are entered according to the University of California's billing codes.

code	data type	code	data type
An	choice of one or more items from a list	L	long integer
Cn	choice of a single item from a list	P	phone number; area codes and extensions handled
D	date	T	text
F	floating-point number	Y	yes/no
I	integer	$	money

modifiers

F	fixed length	U	convert to upper case
K	relational key	V	variable length

Table 3 -- Display/storage formats.

Users conduct three kinds of activities: entering new data, editing already-existing data, and generating reports. In all cases, interaction is via screens and fields supported by editform according to form files. In the former cases, the screen consists of sets of fields which correspond to the stored data items themselves. In the latter case, the screen consists of fields through which the user specifies choices of data items which are to appear in the reports, and fields which specify data extraction criteria (the subset of the data which is actually to appear in the reports.)

Evaluation of IISs

Quantitative evaluation of such systems includes their initial cost, how much it costs to use them, and issues of size and speed -- how much data can be handled, what is the response time, how many users can be supported simultaneously, and so forth. Qualitative evaluation addresses such questions as how difficult they are to use, how useful are the reports which can be generated, and how difficult is it to change the capabilities of the system or add new ones. The remainder of this paper addresses the qualitative issues.

Perhaps the most dramatic proof of the user-friendliness of EDS is that a two-page user manual suffices to explain its complete operation by personnel not trained in computer use. The repertory of commands which a user must master consists of a half-dozen single-keystroke functions. This contributes to EDS's low operating cost.

User-friendliness requires a high level of interaction. Users may, at any time, interrogate the system as to what is going on in general, or what is the format and content of a data field, or what is an example of the expected datum, or what are the available commands. It is hard to get lost.

Validation is an intrinsic part of the data entry process. Data are rejected (with explanatory messages) when they are out of range or logically conflict with other data. This quality control contributes to the accuracy of the reports which are ultimately produced.

In the context of the relational model of data, there are five levels of validation that can be conducted. First, there is the syntactic level. For example, no non-digits can be entered into a numeric field, and phone numbers must adhere to a defined format. The other four levels are semantic. The second level enforces external knowledge. For example, there is no such date as January 32nd; or some

date might be required to precede today's date. The third level involves cross-checking between two or more data items in the currently-entered record. For example, one number may not be allowed to exceed another. Another example would be to entirely disallow entry of data into a certain field depending on the value in some other field; if clinical procedure is not "excision", then do not accept anything in the "method of closure" field. The remaining two levels involve database lookups as part of the verification process. The fourth level cross-checks against other records within the same relation. For example, it may be desired to verify that some value does not exceed the highest such value. Finally, the fifth level of checking checks against other relations. A diagnosis might be accepted into the "found diagnoses" relation only if it is present in the table of diagnosis codes, which is stored as a separate relation. Another example of this level would be to verify that a patient's encounter date does not pre-date his birth date; a datum from the encounter relation is checked against one from the patient relation.

The reports which EDS generates are totally general as to extraction and reporting criteria. Any data element or combination may be used as a selector. For example, a researcher might want a comprehensive display of all males over the age of 40 with a certain disease. An administrator might want to know how many of a certain test were ordered in a given month. Decisions about which reports are desired need not be made as part of the system design process. This would freeze the system's reporting capabilities. Rather, the report-generating procedure is conducted in the same manner as data entry. Reports can be requested on the fly. New ones can be created as they are conceived and as new requirements become evident.

It is a straightforward process to modify screen layouts, data collection, and data presentation to accommodate changing needs. The procedure for accomplishing this is to edit the form files as with any on-line document, and execute some Ingres commands (with Ingres in its standalone mode) to modify storage formats.

Conclusion

The component method can cheaply produce IISs. These systems can easily and at low cost provide administrative and clinical data for health-care providers. We have seen that it is easily possible to take the editform and Ingres tools and apply them to a variety of applications. Furthermore, these tools are compact and can run on small computers, making the tools even more generally applicable.

References

[1] Wasserman, A.I. and D.T. Shewmake, "Rapid Prototyping of Interactive Information Systems", Proceedings 2nd SIGSOFT Symposium -- Workshop on Rapid Prototyping, Columbia, MD, April, 1982.

[2] Wasserman, A.I. and S.K. Stinson, "A Specification Method for Interactive Information Systems," Proceedings -- Specifications of Reliable Software, IEEE Computer Society, Cambridge, MA, 1978, pp. 68-79.

[3] DeRemer, F. and H. Kron, "Programming-in-the-Large Versus Programming-in-the-Small", IEEE Transactions on Software Engineering, SE-2, 2, pp. 80-86.

[4] Parnas, D.L., "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, 15, 12, pp. 1053-1058.

[5] Ritchie, D.M. and K. Thompson, "The Unix Time-Sharing System," Communications of the ACM, 17, 7, pp. 375-385.

[6] Stonebraker, M., E. Wong, and P. Kreps, "The Design and Implementation of INGRES", ACM Transactions on Database Systems, DB-1, 3, pp. 189-222.

[7] Brown, C.D. MD, ed., SNODERM. Baltimore: Waverly Press, 1978.