Structured application generation using XDB

by S. BING YAO and H. KITAGAWA*

University of Maryland
College Park, Maryland

ABSTRACT

A new approach for application system generation, based on the XDB application generator, is proposed. Combining functions of a relational database management system, interactive form processor, and sophisticated report writer, an approach that is based on a formal application model is developed. Advantages of such an approach include structured architecture, incremental development, dynamic prototyping, and high system modularity.

*H. Kitagawa is on leave from NEC Corporation, Tokyo, Japan.
1. INTRODUCTION

One of the most important problems involved in information system development is how to efficiently realize the required system functions. This topic has been studied for a long time under the central theme of software engineering. A considerable number of engineering disciplines have been proposed, and some of them are used in practical software projects. They include software development methodologies (e.g., top-down design and information hiding), specification techniques and tools (e.g., requirements/design specification languages and their processors, and high-level programming languages), development environments (e.g., programmer's work bench, and programming environments), and management disciplines (e.g., configuration control, software metrics, and cost estimation). 1-7

Nevertheless, problems in software development, particularly in the development of application systems, are far from resolved. Many enterprises still encounter difficulties such as programs that do not meet the initial requirements, poor handling of requirements changes, and backlogs of underdeveloped systems. The recent widespread use of microcomputers in business makes the situation even worse. 8 It brought about a tremendous increase in the number of computer utilization requirements, and with it the inevitability that non-professional programmers would have to develop application systems.

Recent attention has focused on a new approach for application system development under the central theme of software engineering. A considerable number of engineering disciplines have been proposed, and some of them are used in practical software projects. They include software development methodologies (e.g., top-down design and information hiding), specification techniques and tools (e.g., requirements/design specification languages and their processors, and high-level programming languages), development environments (e.g., programmer's work bench, and programming environments), and management disciplines (e.g., configuration control, software metrics, and cost estimation). 1-7

Nevertheless, problems in software development, particularly in the development of application systems, are far from resolved. Many enterprises still encounter difficulties such as programs that do not meet the initial requirements, poor handling of requirements changes, and backlogs of underdeveloped systems. The recent widespread use of microcomputers in business makes the situation even worse. 8 It brought about a tremendous increase in the number of computer utilization requirements, and with it the inevitability that non-professional programmers would have to develop application systems.

In this section, we define a model of application systems under consideration within this context. This model specifies the range of application systems which can be generated with application generator techniques, the production of application systems is done without procedural coding and lengthy testing. Several application generators are available at present. They are classified from the following independent viewpoints.

1. Cross vs. resident target. A cross application generator generates application systems to be executed in a different target environment. This includes a different operating system or computer. Application systems generated by a resident application generator are to be executed in the same target environments.

2. Batch vs. interactive specification. A batch application generator generates application systems from specifications written in a specification language. This is similar in a way to the compilation of source programs into object codes. By contrast, an interactive application generator obtains system specification interactively through a dialogue of user-system interaction.

3. Program vs. interpretive code generation. A program generator generates source or object programs from application system specifications. An interpretive generator generates internal code or tables which are interpreted during execution.

An important advantage of application generators is that they can be applied to improve the software development life cycle. Application generators can be used for rapidly prototyping application systems. 9-14 The prototype can be gradually augmented into the final system, or abandoned and the system developed in a conventional style. In either case, rapid prototyping is a very important technique for identifying and resolving problems in traditional software development.

In this paper, we discuss an approach for application generation using XDB. 15-17 XDB is an extended relational database system augmented by a set of application generation tools. In our classification, XDB is a resident, interactive, and interpretive application generator. It facilitates application generation through functions for relational database management, interactive form processing, sophisticated report generation, and automatic menu generation. XDB also enables application development based on a well-defined application system model. The model provides a basis for structured system design, and modeled components are directly generated with tools in XDB.

In Section 2, we describe the application system model. In Section 3, we discuss XDB functions which support the development of application systems representable within this model. XDB examples are given to illustrate the application generation process. Section 4 provides observations by comparing this approach with more conventional approaches. A summary is given in Section 5.

2. APPLICATION SYSTEM MODELING

In this section, we define a model of application systems under consideration within this context. This model specifies the range of application systems which can be generated with XDB.

As several studies in software engineering have clarified, an application system can be modeled from several viewpoints including functional hierarchy, state transition, and data flow. 18 In our model, an application system is first specified from the viewpoint of functional hierarchy. This corresponds to the hierarchical decomposition of the target system. A tree of functional components called activities results from this analysis (for example, see Figure 1). Upper level activities correspond to more abstract functions and lower level activities correspond to more primitive functions. In the decomposition here, leaf activities are to be represented as networks.
of transactions called steps (see Figure 2). Typical data manipulations involved in steps are data entry, database browsing, query, database update, and report generation. We now define the model in more precise terms.

An application system \( (ap) \) is modeled as the following tuple:

\[
ap = (t, D, tf, db),
\]

where \( t \) is a functional hierarchy called an activity tree, denoted by \( t = (A, de) \), \( A \) is a set of functional modules called activities, and \( de: A \rightarrow 2^A \) is an activity decomposition function meeting the tree condition defined below. An activity \( a_i \in A \) is called a unit activity if \( de(a_i) = \emptyset \). \( D \) is a set of activity diagrams, \( tf: U \rightarrow D \), where \( U \subseteq A \) is the set of unit activities, is a function associating an activity diagram with each unit activity, and \( db \) is an application database.

Tree condition is defined as follows. Given \((A, de)\), a decomposition graph can be created by representing each activity \( a_i \in A \) by a node \( n(a_i) \) and drawing a directed arc from \( n(a_i) \) to \( n(a_j) \) if \( a_j \in de(a_i) \). If the decomposition graph forms a tree, \((A, de)\) meets the tree condition.

Using the above formalism, we can represent a functional structure of an application system in an activity tree and associate a collection of primitive program modules with each unit activity. The control flow among the primitive program modules is represented in an activity diagram associated with it. Application menus are modeled as activity tree nodes.

**Example 1**

Figure 1 shows a sample decomposition graph of an activity tree where

\[
\begin{align*}
A &= \{ \text{Order Processing, Customer Information Management, \ldots, Sales Report, Account Report} \}; \\
de(\text{Order Processing}) &= \{ \text{Customer Information Management, Order Taking, Shipping, Payment, Report} \}; \\
de(\text{Customer Information Management}) &= \{ \text{New Customer, Delete Customer, Change Information} \}; \\
\text{and} \\
de(\text{Order Taking}) &= \ldots, \text{etc.}
\end{align*}
\]

Note that a unit activity is represented by a leaf node in the decomposition graph. Here, we further define an activity diagram associated with a unit activity. An activity diagram \( d_i \in D \) represents the control flow within a unit activity and is defined as the following tuple:

\[
d_i = (S_i, L_i, C_i, s_i, d_f),
\]

where \( S_i \) is a set of primitive modules called steps, \( L_i \) is a set of links, \( C_i \) is a set of conditions, \( s_i \) is an element of \( S_i \) called a start step, and \( d_f \) is the following function associating two steps and a condition with a link,

\[
d_f: L_i \rightarrow S_i \times S_i \times C_i.
\]

An activity diagram can be represented in a graph in which nodes, directed arcs, and arc labels correspond to steps, links,
and conditions, respectively. Each step \( s_i \in S \) corresponds to a primitive program module which, when executed, returns a result value. (The result value is similar to the return value of a procedure call, but steps are higher level objects than conventional procedures, and the result value is not restricted to a single value as in the return value of a procedure call.) Conditions to be associated with links by \( df \) are assertions on result values. When the activity diagram \( d_i \) is activated, start step \( s_{i_0} \) is first executed. After execution of step \( s_{i_0} \), the condition associated with every link joining out of \( s_{i_0} \) is evaluated. Suppose the condition associated with link \( l_{i_0} \) such that \( df(l_{i_0}) = (s_{i_0}, s_{i_1}, c_{i_0}) \) is satisfied. Then the control is transferred to \( s_{i_1} \) and \( s_{i_0} \) is executed. A special DEFAULT condition is qualified only if none of the other conditions are satisfied. In this context, we assume the control transfer among steps is single thread and deterministic. In other words, we do not allow cases where more than two conditions are satisfied after the execution of a single step \( s_{i_j} \). If no conditions are met, execution of steps in \( d_i \) are terminated and \( d_i \) is inactivated. Working memory is provided during the activation of \( d_i \). Steps in \( d_i \) can exchange intermediate results through the use of the working memory and the database.

Example 2

Figure 2 illustrates an example of an activity diagram where

\[
S = \{ \text{Customer Identification, Customer Balance Report, Order Entry} \}; \\
L = \{L1, L2\}; \\
C = \{ \text{BALANCE_CHECK_OK, DEFAULT} \}; \\
s_m = \{ \text{Customer Identification} \}; \\
df(L1) = \{ \text{Customer Identification, Order Entry, BALANCE_CHECK_OK} \}; \text{and} \\
df(L2) = \{ \text{Customer Identification, Customer Balance Report, DEFAULT} \}.
\]

This activity diagram corresponds to the unit activity Order Taking in Figure 1. It has three steps. Customer Identification is a start step and is executed first. If the result value of the execution of this step is BALANCE_CHECK_OK, then the Order Entry step is executed. Otherwise, the Customer Balance Report step is executed after the execution of the start step.

So far, we have not specified detailed data manipulation performed in each step. In the context of XDB, application systems are developed around database manipulation and form-based interactive man-machine dialogue. The types of steps can be classified as follows.

1. Form processing
2. Report generation
3. Graph generation
4. Execution of a user's own program

In Figure 2, form processing steps (i.e., Customer Identification and Order Entry) are represented by rectangular symbols, and the report generation step (i.e., Customer Balance Report) is represented by a print-out symbol. We now further define each of these types of steps.

Step 1: Form processing. Form processing steps require interactive data manipulation utilizing formatted display templates on the screen. Forms are electronic version of paper forms so that they are user friendly in business data processing. Extensive research has been done using forms as the major means for man-machine interaction.\(^{17,19-29}\) Most operations required in business application systems such as data entry, data validation, database browsing, query, and database update are specified within the framework of form processing steps.

A form processing step \((fp)\) is defined as follows:

\[
fp = (t, C = BC \cup DC, V, D, sv, sd, P),
\]

where \( t \) is a form template which determines the visual format of the form, \( C \) is a set of cells for entering and displaying data items, \( V \) is a set of validation specifications, \( D \) is a set of derivation specifications, \( sv: BC \rightarrow V \) and \( sd: DC \rightarrow D \) are onto functions, and \( P \) is a set of post-processing specifications. Cells are embedded into the form template. They may be defined either one by one as independent fields, or in a group constructing a table (with "repeating groups") or matrix. Cells are classified into mutually disjoint subsets, namely basic cells (BS) and derived cells (DS). They differ from each other as follows:

1. Basic Cells: Data items for basic cells are interactively entered by the user. Rules for validating input data items are given as validation specifications \( V \) and are associated with basic cells by function \( sv \).
2. Derived Cells: Data items for derived cells are automatically retrieved and/or calculated from data in other cells, the database, and the working memory. For example, values for derived cells may be derived by the system function to display today's date, looked-up from the database, or calculated from other cell values. Rules for deriving the data are given as derivation specifications \( D \) and are associated with derived cells by function \( sd \).

Validation specifications \( V \) can include the following constraints on input data items:

1. Type check
2. Value range check
3. Semantic consistency with other data values in
   (a) cells for the form processing step
   (b) the working memory
   (c) the database

Derivation specifications \( D \) give logic for deterministically deriving data values from other data values in the database, other cells, and the working memory using mathematical calculations, database look-ups, and some system functions (e.g., the function to get today's date). Post-processing specifications \( P \) specify operations performed when the user indicates the commitment for a form processing step. Normally, this happens after the user enters proper data values for all basic cells. Post-processing specifications \( P \) can include the following operations:
CUSTOMER IDENTIFICATION

CUSTOMER NUMBER:  [cno]  
NAME:  [cname]  
ADDRESS:  [caddr]  
CITY:  [ccity]  
PHONE:  [cphon]  
CLASS:  [cclass]  
BALANCE:  [balance]  

BALANCE CHECK OK:  [balance_check_ok]  

Figure 3—Customer identification

1. Computation of the result value
2. Write to the working memory
3. Update of the database

Data items displayed in cells are discarded after the form processing step is committed. Therefore, unless they are explicitly stored in the database or the working memory during post-processing, data entered by the user for basic cells are lost forever. (The system may provide a "form base" to directly store the complete data on forms. However, to simplify the model described here, we do not assume it in this context.)

Example 3

Figure 3 shows an example of a simple form template which might be used for the form processing step Customer Identification. Cells are represented by brackets. Suppose we have the customer information in the application database representable as the following table:

CUSTOMER (cno, cname, caddr, ccity, cphon, cclass, balance).

We can then designate cell "cno" as a basic cell, and make the others derived cells. That is to say,

BC = {cno}
DC = {cname, caddr, ccity, cphon, cclass, balance, balance_check_ok}

The validation specification for cell "cno" gives valid range of the data inputed for "cno" as well as its data type. It will also assert that the input value has to match exactly one occurrence for the attribute "cno" in the CUSTOMER table. The derivation specifications for the derived cells except "balance_check_ok" are basically queries to the table. The given "cno" value will be used for the key in the queries. The derivation specification for "balance_check_ok" will be a simple if-then-else logic. For example,

if class = 'A' and balance < LIMIT_FOR_A or class = 'B' and balance < LIMIT_FOR_B or class = 'C' and balance < LIMIT_FOR_C or then balance_check_ok = 'OK'
else balance_check_ok = 'NO'.

Post-processing specifications will include two commands; the first gives a computation rule for the result value of this form processing step. It might be formulated as the following logic:

if balance_check_ok = 'OK'  
then RESULT = 'BALANCE_CHECK_OK'
else RESULT = 'ERROR'.

The second command will be to write the current "cno" into the working memory provided for the activation of the Order Taking unit activity. This value is referred to in the execution of either Order Entry step or Customer Balance Report step to find out the "cno" of current concern.

Step 2: Report generation. Report generation is another important step in an activity. Essential data handling in a report generation step includes data retrieval from the database, calculation of derived data values, and formatted display and/or print-out. A form processing step can also formulate data retrieval, calculation, and formatted output. However, one of the major differences between form processing and report generation is that the former includes interactive data input from the user, while the latter does not. Another difference is that a small amount of data on a sheet of paper is manipulated in the execution of a form processing step. By contrast, the handling of voluminous data is not unusual in report generation.

Step 3: Graph generation. In general, graph generation is included in the same category as report generation. As in report generation, graph generation involves retrieval of relevant data, calculation, and formatted output of the result. However, the output is given as bar charts, line charts, pie charts, scatter plots, and so on.

Step 4: User program. Most business data processing can be formulated as form processing and report generation (including graph generation). However, since there are a variety of applications, in some cases all the user requirements for data processing cannot be met with the above three classes of steps. In such cases, the user can define a step as the execution of his own program. Of course, the user is provided with primitive program modules such as database interface routines to access the database.

3. APPLICATION SYSTEM DEVELOPMENT

As discussed in Section 1, XDB is a resident, interactive, and interpretive application generator. The architecture of XDB is shown in Figure 4. The use of XDB facilities is divided into two phases, application system generation and application system execution. In the former phase, XDB generates applica-
tion system descriptions from the specifications given by the user. The user is provided with several interactive modules for activity tree definition and step definition. The application system descriptions are interpretively executed in XDB environments during the application system execution phase. Several types of interpreter modules are provided for the execution phase.

As the core for application development, XDB has the XQL relational database management system. XQL's functions are based upon the specifications for the SQL/DS database management system. In the XQL environments, XDB supports the interactive, evolutionary development of application systems. The following discussion shows how each aspect of the application system modeled in Section 2 is handled in XDB. We will use a small order processing system as an example. Suppose the requirements analysis and database design have already been done for the example system. We can define the following four tables.

CUSTOMER (cno, cname, caddr, ccity, cphon, cclass, balance)
ORDER (ono, odate, cno, sname, saddr, scity, sphon, sdate, pdate)
PART-ORDER (ono, pno, pqty)
PART (pno, pname, pdesc, pprice, pqoh)

We will show how this kind of application system is interactively designed and generated.
The development is performed in a top-down fashion. First, the activity tree is designed to determine the functional hierarchy of the target system. For this example, the activity tree as shown in Figure 1 is defined. In XOB environments, the activity tree is mapped to a hierarchical menu, and defined with the menu writer. Figure 5 shows a session example of the menu generation. No specification language is used for defining the menu. Instead, the definition is performed step-by-step through a number of interactive sessions. In these sessions, a set of commands can be used to specify menu screens corresponding to non-leaf nodes in the activity tree. The definition is completed through the command specification and the input of supplementary information, if required.

In the execution phase, the menu runner presents the menus. The menu screens defined in the generation phase are used to traverse the hierarchy and execute data processing operations formulated as unit activities. Figure 6 shows the main menu of the example order taking system. This menu corresponds to the root activity in the activity tree shown in Figure 1. When unit activities such as Order Taking and Shipping are selected, the steps defined in them are executed according to the flow to be specified in the activity diagrams. When other activities are selected from the main menu, a sub-menu is displayed on the screen.

Activity diagram

The activity diagram determines the control flow within a unit activity. With the use of the activity diagram writer, the user can interactively define the control flow structure. When the control structure is complicated, a visual aid with which the user can directly edit the activity diagrams becomes indispensable for performing the definition. Presently, XDB represents the control as a sequence of steps with all links tagged with condition NOT 'ERROR.' The definition is thus a linear list. If fatal errors occur in each step, the result value is automatically set to 'ERROR.' The result value 'ERROR' is also returned when the step is aborted upon user request or is set in post-processing.

The definitions of step sequences are interpreted by the activity diagram runner to control the execution of steps in the execution phase. Suppose the activity diagram shown in Figure 2 is slightly modified here, and the sequence of steps Order Entry and Customer Balance Report is defined for the unit activity Order Taking. If Order Taking is selected from the main menu in Figure 6, then the step Order Entry is first executed. If its resultant value is 'ERROR', the execution of the activity is terminated. Otherwise, Customer Balance Report is executed as the next step to Order Entry.

Step

As mentioned in Section 2, four classes of processing are allowed for steps. In the above example of the activity Order Taking, Order Entry is a form processing step, and Customer Balance Report is a report generation step.

Form Processing. Form processing steps are interactively defined with the form writer. To define it, the user first designs the form template on the screen. In so doing, he also defines cells embedded in it. Cell type (i.e., basic/derived), data domain (i.e., integer/real/string/money/date), and display format information is associated with each cell. For basic cells, the user can give the validation specification, for example, indicating the range of input data values. For derived cells, he has to specify the derivation specification. The derivation specification is an expression combining arithmetic expressions and XQL query statements.

The form shown in Figure 7 will be used for Order Entry. As shown in this example, forms having complicated data
structures (e.g., repeating groups) can be defined in XDB. Here “custno,” “pt.x,” “qt.x,” “shname,” “shaddr,” etc. are basic cells, and values for them are inputed by the user. Validation specifications could be given to assure the semantic constraints for maintaining proper balance values. Data values for cells “custname,” “custaddr,” “custcity,” and “custphon” will be retrieved from the database. Their derivation specifications will be given as the following XQL query:

```xql
select cname, caddr, ccity, cphoIi
from CUSTOMER
where cno = [cno],
```

where “[cno]” means the input data for the cell “cno.” Similarly, data values for “it.x” and “pr.x” for each table entry will be retrieved from the PART table. Derivation specifications for cells “am.x,” “sub_total,” “total,” etc. will be based on arithmetic calculations from values in relevant cells. Data value for the cell “idata” for today’s date will be derived by the built-in system function.

The post-processing specifications for this form processing will include storage requests of new records to tables ORDER and PART_ORDER. They are also specified using XQL database update statements. The balance value in the CUSTOMER table will also be updated during the post-processing.

Form execution is performed by the form runner. In the execution phase, data values for derived cells are obtained in a data-driven way; that is, their derivations are triggered as soon as their source data become available. Therefore, values for “sub_tot” and “total,” for example, are automatically recalculated when a new data entry is given to the source cells involved with its computation. Data stored in the database is retrieved by directly executing the specified XQL statements embedded in the derivation specification. Commands in the post-processing specifications are executed in a similar way when the user indicates the commitment by pressing a proper function key.

**Report generation.** Report generation steps are also defined interactively. In the generation phase, the user first formulates XQL queries and executes them to retrieve the data. In the report generation step, Customer Balance Report, the following XQL query will be used to retrieve the relevant data.
490 National Computer Conference, 1985

select cno, cname, caddir, ccity, cphon, balance
from CUSTOMER, ORDER
where CUSTOMER.cno < 6 and
CUSTOMER.cno = ORDER.cno and
odate < 1/1/84

The data is first displayed in a flat table on the screen, then the user interactively issues editing commands and formats the report into a desirable form. These are performed under the control of the report writer. The report writer automatically generates a description file that logs all of the interactive commands entered by the user. In the execution phase, the report runner interpretively executes the description to generate a fresh report from the current data stored in the database. Figure 8 shows an example report generated in the Customer Balance Report step.

Graph generation. Graph generation steps are defined and executed similarly to report generation, but differs in that the query results are shown in several types of graphs. XDB provides facilities for generating pie charts, bar charts, line charts, scatter charts and x-y plot.

User program. In principle, any program can be executed in steps in XDB environments to meet a variety of application system requirements. It can be a user-coded program or system program such as editor or utility programs. User-coded programs may be coded in languages such as C, BASIC, and Assembler. Typical examples of such user programs manipulate the database and are written in C. XDB provides the library for C programming language and facilitates the development of such programs. It includes basic functions for XQL relational database access and update. The return values of user programs become result values of the steps in which they are executed. User programs are pre-compiled outside XDB, and their object programs are incorporated in XDB environments under the external program call definition module. Their calls in the execution phase are handled by the external program call handler.

4. ADVANTAGES OF THE XDB APPROACH

In Section 3, we explained application development in XDB. We now summarize the advantages of our approach over other application system development approaches.

Well-defined Application Architecture

Our approach provides a simple, yet powerful application model. The development along the model is directly supported by the utilities in XDB. The application system architecture is composed of well-defined components and is very modular. Therefore, the generated application system is easy to understand and has good maintainability. The development process can also be well-structured, since the target of the efforts is well specified.

Incremental Development

In the XDB environments, an application system can be developed incrementally from the prototype. For example, when only the activity tree is defined, the developer can evaluate the "shell" design by executing the menu handler. This will show how the hierarchical menus are presented to the user. After the evaluation and modification of the activity tree, the developer can proceed to the design of an activity diagram for each unit activity. In this way, the developer can gradually refine the prototype through the cycle of design and evaluation to obtain the final application system.

Dynamic Prototyping

Although the development can be gradually expanded from the core prototype, it need not follow a predefined specific development procedure. In some cases, the design of a specific step is of great importance and has an influence on the global architecture of an application system. In such cases, prototyping of the step of specific concern can be done even though the developer does not define the activity tree or activity diagrams. In this way, prototyping can be performed dynamically.

Modularity

Even though the development is performed incrementally and dynamically, modularity of the target system is always ensured within the XDB environments. The development process is performed in a top-down fashion, and the designer is guided into a structured design procedure which results in a more modular system. The target system is easy to test and modify.
Non-procedural Specification

The specifications of the target application system are interactively given by the developer in a non-procedural way. This not only facilitates the development, but enhances the maintainability of the target system because the specifications are given in high level descriptions. To modify or change the system, an analyst can interactively modify the design specification, and the system is automatically modified without the need for "system regeneration." Automatic generation of system documents can also be accomplished easily because the high level system descriptions are stored in the system in a well-organized way.

5. SUMMARY

In this paper, application generation with XDB is discussed. We proposed an application system model defining a class of applications under the XDB environments. Then, we discussed how each modeled aspect of the application system can be developed by XDB facilities. The discussion follows an example for a small order processing system. Finally, we summarized advantages of our approach over other system development approaches. The current version of XDB is a proprietary product implemented by Software Systems Technology on IBM personal computers. Versions that run on the generic MS/DOS and UNIX environments are also being developed. The high modularity of the system enables it to run with only 192K bytes of memory. The power and usefulness of the system have been verified through a number of experimental uses of XDB in realistic, personal-computer-based business application systems.

ACKNOWLEDGMENT

The authors are grateful to Mr. O. Q. Wu for reading and commenting on the draft of this paper.

REFERENCES
