Apr 1st, 8:00 AM

The Express Project: Automating the Software Development Process

John W. McInroy

Leonard S. Baurnel

The Express Project: Automating the Software Development Process

By

Dr. John W. McInroy and Leonard S. Baumel

Lockheed Software Technology Center
2100 East St. Elmo Road, B/30E
Austin, Texas 78744
512/448-9751

The Express Project: Automating the Software Development Process

Dr. John W. McInroy and Leonard S. Baumel

Abstract

The goal of the Express project is to provide an efficient software development environment for embedded systems. The approach is to create a new life-cycle paradigm, using rapid prototyping to validate system specifications. The rapid prototype will be made possible by using (1) very high level specification languages that automatically generate code and (2) a user-machine interface that helps both in the layout of the design and in the specification of the input devices and output screens for the embedded system.

All user interactions with Express are integrated through Express's knowledge-based Framework, which will support efficient, interdisciplinary communications. The Framework is designed to support evolutionary prototyping. The high-level view of the embedded system can be created and evolved concurrently with the low-level specifications of processing segments that are understood and known to be required. System engineers will define the high-level view (including allocation of requirements from layer to layer), and specialists will create low-level diagrams and specifications for processing threads in each of their (initially) independent areas. In small steps the two views will be merged into a single architecture diagram. One can "zoom in" on one subsystem and be presented with the fine structure of the subsystem down to the level of executable specifications.

A second subsystem, which is reached through the knowledge-based Framework, is the Graphical Specification Subsystem (GSS) for Displays. It will make human-machine interface engineers more productive when designing operator displays for embedded systems. It will allow them to "build" a display graphically. They can select icons from a menu, position and size each instance of an icon graphically (by mouse action), and specify in a natural way the desired interaction with other portions of the embedded system. Gauges, graphs, and maps are examples of objects represented by icons. The GSS also will be used to specify simulated input devices to the system, such as mice, push buttons, and joysticks.

The Express Project: Automating the Software Development Process

By Dr. John W. McInroy and Leonard S. Baumel

Lockheed Software Technology Center, B/30E
2100 East St. Elmo Road, Austin, Texas 78744
512/448-9751

WHAT IS EXPRESS?

Express is a very high productivity software development environment. It provides the following capabilities:

- Automatic code synthesis from specifications
- "Domain-natural" specification languages for embedded systems
  - –Graphical specifications for end-user displays
  - –Objects and policy specifications
  - –Symbolic mathematics
  - –Architecture diagrams
- A framework for integrating and executing specifications

Express offers a new paradigm for software development:

- Concurrent life cycle phases
- Rapid prototyping
- Early end-user involvement
- Validation by executing prototypes
- Maintenance from the specification

High productivity is attained by automatically synthesizing high-order language code from specifications. The higher the level at which the user can specify the system, the less writing or coding the user has to do, and the more efficient the process. Express uses knowledge-based specification languages, which capture programming expertise in particular domains to provide this efficiency leverage. The "domain-natural" specification languages of Express fall into four categories: graphical specifications, objects and policy specifications, symbolic mathematics specifications, and architecture diagrams:

- The Graphical Specification Subsystem (GSS) for Displays language allows the designer to develop the human interface by selecting the appropriate icons from a menu and placing the icons on the screen at the desired locations via mouse action. Each icon can be sized, moved, and combined with other icons to develop the output screen as well as the input device simulation on that screen.
- The Objects and Policy (O&P) specification language allows the user to define the objects in the domain and the policies governing the behavior of those objects. The designer provides the specifications via a mixture of filled-in forms and free-form text in constrained natural language. This object-oriented style of programming allows a very declarative style of specifications and maximizes the productivity leverage.
- The Symbolic Mathematics specification language are based on the Macsyma system for symbolically manipulating algebraic expressions and, for those domains that are mathematical in nature, provide the natural specification language to dramatically increase productivity.
- The Architecture Diagram specification language supports systems engineers in specifying the control structure of module interconnnections by providing a graphical interface with which to design the system.

The Express environment has a framework system, which hosts the Architecture Diagrams and manages specifications in the three other specification languages. The Framework allows combinations of all specification languages. It provides integration at the knowledge-base level and execution control in run-time mode. The Framework is used to represent the design of the embedded computer system and provides the top-level user interface to Express.

Express introduces a new paradigm for software development. It compresses the sequential steps of the waterfall model into a series of concurrent efforts, producing rapid prototypes of the system. The productivity leverage of the specification languages listed above makes the generation of the prototypes practical. These prototypes allow early end-user involvement in the design of the final product, increasing customer satisfaction when the product is delivered. The running prototype validates the specification, and the system is then maintained at the specification level. Code is synthesized by the knowledge-based compilers of the specification languages.

Express Specification Classes and Framework

Figure 1 illustrates the top level of the user interface to Express. When logging onto the system, the user will be placed in the Framework system, shown in the bottom screen face. Through specific mouse actions and menu choices, the user creates an architecture diagram to serve as the main user interface to Express. The architecture diagram on the bottom-row screen of figure 1 shows the top-level functional allocation of the system. Each display in the second row shows the finer structure of one particular top-level entity. This zoom-in view is selected by an appropriate mouse click while the mouse is pointing to the entity to be examined in finer detail. Several levels of the allocation will exist in the final embedded system architecture. As the design entities are modularized down, the final step is called the primitive entity and represents the functionality that will be specified in one of the domain-natural specification languages previously discussed. The top row contains examples of three of the specification languages, with the control structure shown in the architecture diagrams constituting the fourth class of specification.

The Objects and Policy entity is generated by a series of menu choices that define the objects in the domain and the behavior of those objects, which is governed by the policy statements about each object. The Graphical Specification of Human Interface entity represents the second major class of user interface in Express; that is, the definition of the user interface to the embedded computer system being developed with Express. Two user interfaces are of interest here: the user interface to

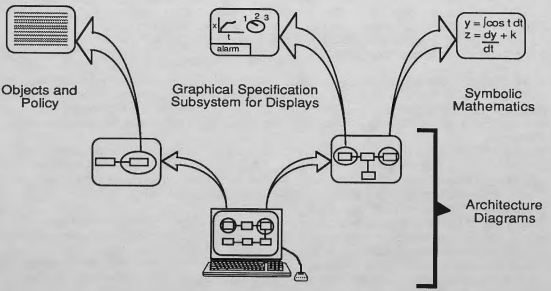

Fig. 1. Express specification classes and framework

the designer, a user of Express, and the user interface to the end user of the embedded system. We think it will enhance the embedded-system user interface if its designer has the tools to allow interaction with a prototype of the embedded system. The Symbolic Mathematics entity is a specification in a mathematical domain. This specification class allows the user to express mathematical concepts in the mathematical notation that is natural to the problem and to the specifier.

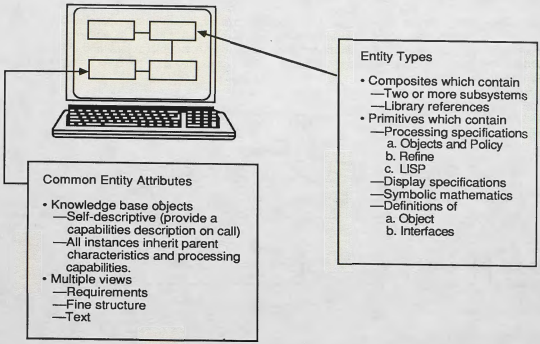

Fig. 2. Framework entity description

Framework Entity Description

The boxes which appear on the architecture diagrams of the framework screen in figure 2 are knowledge-based entities, which can be of two basic types. The top-level entities are *composite* types, composed of two or more other entitites, which themselves may be composite entities or *primitives*. The composite entities may also contain library references, allowing Express to make use of preexisting software packages when available. The second type of entity is the primitive, which contains the machine-processable, knowledge-based, domain-natural specification language description of an embedded computer system module.

The processing specifications will be in the Objects and Policy (O&P) language described earlier. The O&P languages are built upon the Refine specification language provided by Reasoning Systems. These primitives may also contain Refine code, or even LISP for those cases where a suitable higher-level language does not exist. The display specification primitives contain the Graphical Specification icon-built representations of the output display screens and input devices for the embedded computer system being developed. The Symbolic Mathematics primitives contain the mathematical specifications describing the functional modules, which are naturally described in mathematical notation. The definitions of the object and interface primitives define the knowledge base objects in the domain of the embedded computer system.

Since each entity is a knowledge base object, the entities all share certain attributes. The entity is self-descriptive and will return a description of itself when requested to do so. Since lower-level composite entities are built from higher-level entities, the characteristics of the higher-level entity

are transmitted to the lower-level entity via the inheritance mechanism. The knowledge-base nature of the entity allows the user to access multiple views of the entity. For example, the user may view the requirements associated with each entity, along with the requirements traceability. Also available are the fine structure and test views of the individual entities and of the overall design of the embedded computer system.

Object Inheritance

Figure 3 shows subclasses of the general screen object defined by the Graphical Specification Subsystem (GSS) for Displays and illustrates the concept of inheritance as one moves to more-specific object definitions. Each item in this figure represents an object class, and when, for example, a dial gauge icon is selected from the menu of icons and placed on the screen, a specific instance of dial gauge is created and can be linked to an application parameter via the underlying knowledge base.

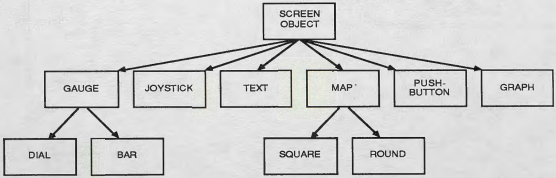

Fig. 3. Object inheritance

Express Architecture

Figure 4 shows an abbreviated overview of the Express architecture. The Executive/Framework shown at the top is the primary user interface to the Express development environment. The Graphical Specification for Displays of the application user interface is the subsystem that is provided by Express to give the designer the tools needed to specify and prototype the embedded system user interface. This subsystem will be the focus of the remainder of this paper.

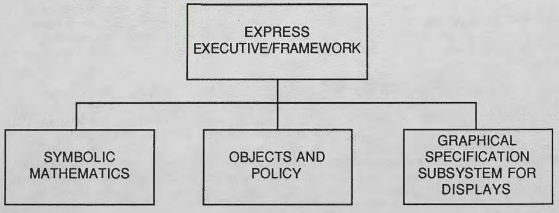

Fig. 4. Express architecture

The rest of this paper focuses on ways in which a designer of a human interface specifies that human interface. These specifications are input to the Express environment (fig. 5). The resulting output from the Express environment is executable.

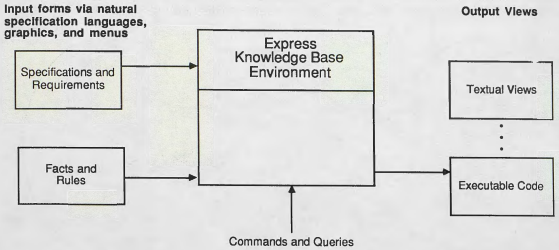

Fig. 5. Express inputs and outputs

GRAPHICAL SPECIFICATION OF HUMAN INTERFACE

Graphical specifications and the Express subsystem that supports them are one of our four research efforts with respect to specification languages, in addition to architecture diagrams, objects and policy specifications, and symbolic mathematics.

Objectives

The major objective is for a designer to be able to specify the human interface for a target system in a natural way; specifically, to be able to specify it by laying out the picture that the end user of the embedded system will see. (We expect that modern user interfaces are pictorial and contain menus, so we provide support for this type of interface, as well as for simulated input devices.) The resulting specification will be executable, first as a rapid prototype, then, after multiple iterations, as a target system. It also is an important objective that Express be easy for the designer to learn, use, and remember.

Approach

The approach is object-oriented and knowledge-based, and it employs direct manipulation. When users learn how to use one type of object, they will be able to predict how to use other types of objects. Objects are represented pictorially, but there is more to an object than meets the eye, as described below. We use a knowledge-based approach, in that objects contain other knowledge in addition to pictorial information.

We also use a direct-manipulation approach, which means that designers can "get their hands on" objects and control them, by pointing and selecting with the mouse, with respect to layout, size, and so on.

Analysis of Human Interface

We use an analysis of human interface developed by John Bennett at the IBM Research Laboratory, now in Almaden, California.

A human interface consists not only of what a person *sees* (or hears, or otherwise senses), but also of what a user must *know*, in order to use the system, and of what a person can *do* (fig. 6). We refer to what the user sees as a *presentation language*, and what he or she can do as an *action language*. We try to reduce what the user must know, so that the user can focus on the task at hand,rather than on how to use the system. That is, we try to reduce the user's *cognitive load*.

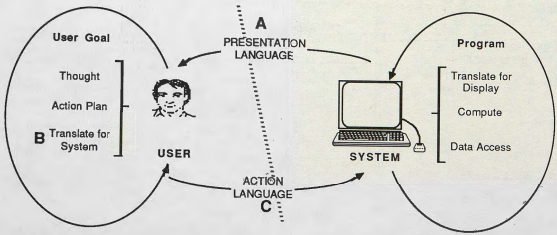

Fig. 6. Analysis of human interface. *A*, what a user *sees* (or senses); *B*, what a user must *know; C*, what a user can *do*.

Express Output: Operating Rapid Prototype

For our independent research review in spring 1987, we built and ran a demonstration that included the Graphical Specification Subsystem for Displays, along with a sample application (fig. 7). The network-support aspect of the Express Framework and run-time Executive allows the user interface to run on a separate Symbolics machine from the application. The designer can sit at a display that simulates the human interface for the target application and can edit both the presentation and the knowledge behind it. At this time, the interface to the application is coded in LISP.

Specification of an Object in a User Interface

An example of a type of object that we provide to the designer is a map (fig. 8). Like other types of objects that we provide to the designer who uses Express, a map contains not only a presentation "layer," but also other "layers," one of which contain knowledge about maps, such as how latitudes and longitudes work. A designer interacts with this layer via menus, so that the designer does not have to code programs in a linear syntax.

A map also has an interface with the rest of the system, for example, with messages that indicate where objects such as aircraft or submarines have been detected. Our group at Texas A&M University is working on a capability to specify this interface without programming.

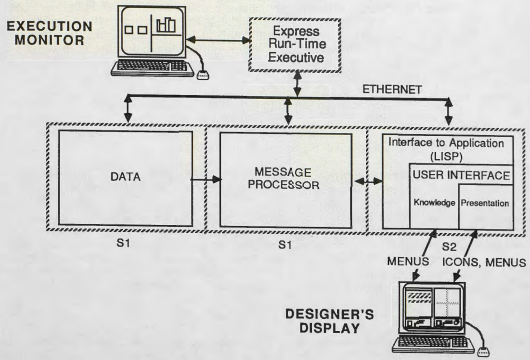

Fig. 7. The output of Express is an operating rapid prototype. S1 and S2 represent Symbolics processors 1–2.

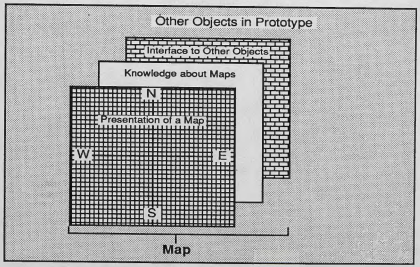

Fig. 8. Specification of an object in a user interface

Prototype of a human interface. The display in a prototype built via Express can simulate not only the end user's display screen, but also controls that may be realized as hardware in the embedded system, such as joysticks and pushbuttons. A given object in the knowledge base of the application, such as a message that an unknown aircraft has been sighted, can be displayed in more than one way—for instance, on a map and in a table.

An example of a human interface that the designer can create is shown in figure 9. In this example, by means of the simulated pushbuttons labeled List and Plot, a user can control whether updates are to be allowed or whether the display is to be "frozen." With other simulated pushbuttons, the user also can control whether to display friendly objects, enemy objects, unknown objects, or some combination. On the map, the user can also specify whether or not to plot the history of the sightings of objects, as well as their current positions.

Specifying an object. In the presentation shown in figure 10, the designer is about to specify a map. The designer has requested this Objects menu by means of mouse clicks. The mouse pointer is pointing at an icon for a map, as indicated by the text at the bottom of the menu, indicating Square Map. The user clicks left to select the Square Map object.

Specifying initial attributes of an object. Figure 11 shows the form that allows a user to provide knowledge about a map, such as the latitudes and longitudes to be displayed on it.

Specifying output by means of an object. To display output from the application on this map, the designer selects the form shown (fig. 12) from a menu. In the current level of this demonstration, the user specifies the name of a function written in LISP. This function maps data from the application to the user interface, where the application was specified more or less independently of that interface. In the research at Texas A&M, we plan to provide a user interface that will allow the designer to provide linkage between applications and user interfaces without coding.

Research Aspects

What's new. As far as we know, the capability to generate executable code from objects such as maps and graphs within a user interface is new. Also, we are not aware of executable specifications for operator displays that can be integrated with other types of executable specifications, as we are doing.

Relevance. Support for objects such as maps and two-dimensional graphs is relevant to aerospace requirements. This research is also relevant in more general terms, in that it supports rapid prototyping, by which we hope to increase customer satisfaction. Also, we hope to enhance productivity by providing a language that is appropriate for the task of specifying human interfaces.

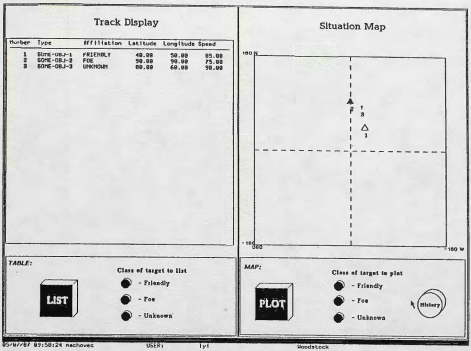

Fig. 9. Prototype of a human interface

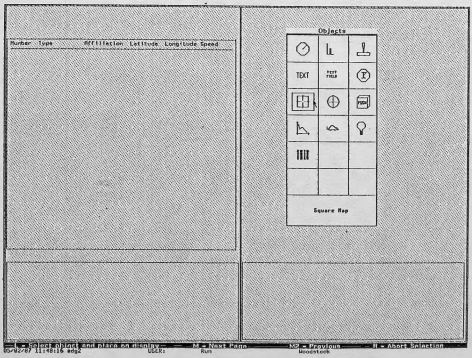

Fig. 10. Specifying a map

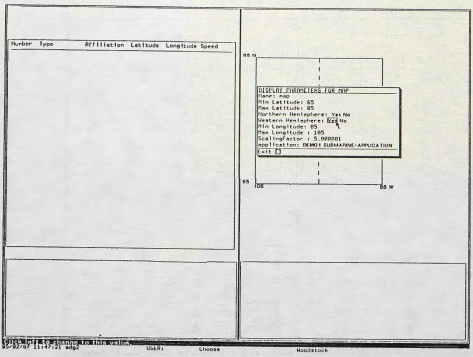

Fig. 11. Specifying initial attributes of a map

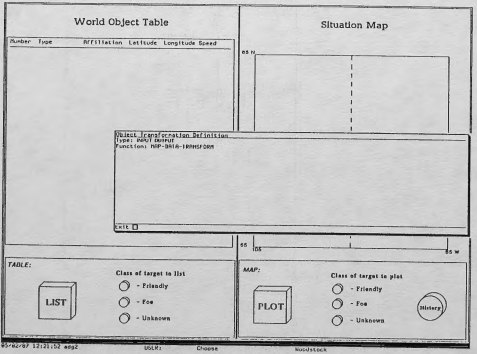

Fig. 12. Specifying output to a map