A C++ SOFTWARE ENVIRONMENT FOR THE DEVELOPMENT OF EMBEDDED SIGNAL PROCESSING SYSTEMS

Joseph M. Winograd and S. Hamid Nawab

ECS Department, Boston University, Boston, MA 02215

ABSTRACT

A new environment for the rapid development of embedded signal processing software is described. The environment encourages incremental design via modular and hierarchical structuring of applications, and additional features are included which support the prototyping, testing, implementation, and integration stages of the system design cycle. Written in C++, the environment is comprised of a scripting language for the definition of system components and a class library which includes a basic application framework. Support is provided for incorporating both numeric and symbolic signal representations, as well as integrating multiple signal processing techniques within a single application. A sophisticated control mechanism allows dynamic scheduling of signal processing operations according to algorithmically defined schema. Signal processing applications developed in this environment are themselves objects, and are suitable for embedding within a larger overall system.

1. INTRODUCTION

This paper describes a new software environment for the rapid development of embedded signal processing systems which offers a unique combination of features with respect to currently available design tools such as Ptolemy [1], ADE [2], and Matlab [3]. The environment presents a unified platform in which embedded signal processing applications which require sophisticated control can be designed, prototyped, tested, and implemented. In particular, this can be accomplished without the need for the labor intensive and error prone steps of manual format conversion and design reentry which are inevitable when incompatible tools are required for different stages of a system's design cycle. An object-oriented design philosophy is employed throughout the environment, enabling applications to be constructed and tested in an incremental manner. The environment has been used to develop systems employing the IPUS signal processing model [4], and has therefore been entitled the IPUS C++ Platform (ICP).

This work was sponsored in part by the Department of the Navy, Office of the Chief of Naval Research, contract number N00014-93-1-0686 as part of the Advanced Research Projects Agency's RASSP program. It was also sponsored in part by the Rome Laboratories of the Air Force Systems Command under contract number F30602-91-C-0038.

2. SOFTWARE DETAILS

ICP has been written using the C++ language and was developed and tested with the GNU g++ compiler on a Sun 670MP platform. C++ was selected for its efficiency, portability, reverse compatibility with the C language, and the potential for porting applications to DSP hardware through the use of target independent C++ to C translators (such as one of the widely used derivatives of AT&T's *cfront* [5]) in conjunction with C language cross-compilers.

ICP provides a library of base classes with a rich set of default behaviors. Mechanisms are provided for all system level features such as execution of control plans, invocation of signal processing algorithms, and organization of shared data objects. The construction of a signal processing application is a process of deriving concrete objects with application specific behaviors from these base classes. The application developer must define signal representations, signal processing algorithms, and application specific control strategies. The resulting signal processing system is itself an object, and is suitable for embedding within a larger overall software system.

A scripting language based on the C++ pre-processor's macro facility has been developed which simplifies the process of constructing an application. Defined by a formal production grammar which may be considered an extension to that of the C++ language, it facilitates the derivation of software components through the use of natural, direct statements. We have found it to enhance code readability, greatly reduce the amount of keyboard entry required for coding a system, and reduce the amount of C++ knowledge required by system implementors. A source code segment using this scripting language is given in Fig. 1(a). Because it is directly parsed and compiled, no additional translation, conversion, or code-generation steps intervene between the high-level system design and the testing and integration stages of development.

Central to the design of an ICP application is the *application* class which serves as the framework within which all other system components are instantiated. The application class provides a basic interface between the signal processing sub-system and external software components. It may be directed to process signals from one or more signal sources and will provide the results of processing along with information about the signal processing system's current internal state. It also provides a trace facility which allows all internal structures and operations to be displayed in textual and graphical form at runtime for performance

0-7803-2431-5/95 $4.00

monitoring and testing. We have used these facilities to implement an X-Windows front-end for the interactive execution and testing of ICP applications.

3. ARCHITECTURAL MODEL

The basic system model employed within ICP is that of a collection of independent signal processing algorithms which are invoked according to algorithmically defined control plans. In addition to supporting standard signal processing architectures, this paradigm allows the development of systems which may alter their processing activities in response to conditions such as fluctuating system resources, requests from other system components, or the results of their own calculations.

This flexibility is provided through the use of a control mechanism (or *planner*) based on the RESUN control paradigm [6]. The planner allows for both strategic (plan-based) and opportunistic (reactive) control to be applied. The strategic component is based on an application specific goal/plan/subgoal hierarchy (referred to collectively as *control plans*) in which all system goals are explicitly submitted to the planner and may be addressed by either a single algorithm or decomposed into an algorithmically defined sequence of further subgoals. Fig. 1(b) shows a schematic diagram of a portion of the control plans for the radar signal analysis application described in section 4. The opportunistic component of control is provided by daemon-like refocus units, which are objects that may be instantiated from any point in the system and are invoked to redirect control flow whenever a specific pre-condition arises in the system.

ICP's architectural model casts the process of generating the output associated with a given input signal as a series of transformations between multiple abstract signal representations [7]. These representations may be strictly numeric, such as raw signal data or the results of a DFT, or symbolic, as in the determination of a radar return cell as background noise, clutter, or target. All signal representations are stored on a global, hierarchical blackboard data structure [8].

4. APPLICATION

We have used ICP to construct a system for the analysis of radar signals based on techniques from [9, 10, 11]. The problem addressed by the system is the detection and separation of clutter patches of homogeneous probability distribution within range-azimuth returns of a non-imaging radar. The input to the application is a record of the amplitude of radar pulse returns across a partitioned range-azimuth plane. Its output is a list of homogeneous patches, each described by their boundary and a characterization of their distribution.

The application incorporates four signal representations, each derived from the ICP signal representation base class. Instances of these objects are maintained in separate areas of the blackboard-structured database which we refer to as the *return*, *map*, *region*, and *patch* levels. The return level representation contains the input to the system—raw radar return data. The map representation contains a binary mapping of each return cell to a label of either *clutter* or *clear*. Each instance of the region representation describes a region of contiguous clutter cells within a map level signal. The patch level representation describes an area of contiguous clutter cells with homogeneous probability distribution and forms the output of the application. Graphical displays of these representations, generated using the trace facility of ICP, are shown in Fig. 2(a)-(d).

Algorithms for producing these successive representations of the radar data have been adapted to a control strategy for signal reprocessing [4] and encoded as a set of control plans. Within these plans, the map representation is produced from the return representation by application of a technique from [9] in which the parameters to a series of morphological operations are iteratively refined until satisfactory segregation of clutter regions from background noise is achieved. Clutter regions are produced from the map representation using a recursive nearest-neighbor region growing algorithm. Patch level representations are produced from the clutter regions through the use of Ozturk's algorithm [10] for distribution estimation. Statistical characterizations are made independently for a series of non-overlapping 10-by-10 cell tiles within each clutter region to detect possible homogeneous patches. The results of that analysis are subjected to a series of practical constraints on patch geometry and reprocessing is performed with alternate tilings to verify and refine patch boundaries.

5. RELATION TO OTHER WORK

In order to distinguish ICP from other tools which might be employed during the development of an embedded signal processing system, we present in Table 1 a list of key design elements of ICP and indicate their presence (or absence) in selected tools from the commercial and research communities. It should be noted that each of these tools provide additional features not available in ICP and that complete evaluations are beyond the scope of this report.

We have already pointed out several benefits of taking an object-oriented approach to the design of complex signal processing systems. The application of this design philosophy to signal processing domains was pioneered by a series of projects at MIT which culminated in the development of the ADE system [2]. A focal point of this work was the development of a well-defined concept of the "signal as object." Related work by Milios and Nawab [7] proposed the use of numeric and symbolic representations for encoding signals in a hierarchy of levels of abstraction, as is done by ICP.

In ADE, the automation of certain design tasks is facilitated by the symbolic encoding of properties of the component systems. This property based analysis provides a higher level of description (using a *metalanguage*) than that provided by an input-output relation or algorithmic description of a system, simplifying ADE's automatic rearrangement of composite systems. A strong analogy may be drawn between this metalanguage and the hierarchical RESUN control structure adopted by ICP. Each algorithm which performs a transformation between signal representations (i.e. a system, in the classical sense) is declared in the scripting metalanguage of ICP to fulfill a specific subgoal in the control plans. This information is used by the planner

Design Tools				System Features
ADE	Matlab	Ptolemy	ICP	
X		X	X	Supports object-oriented design
X		X	X	Abstract signal representations
X		X	X	"Metalanguage" for system description
		X	X	Compiled targets
X	X	X	X	Text-based debugging aids
	X	X	X	Graphical debugging aids
		X	X	Dynamic scheduling
			X	Strategic and reactive control

Table 1: Comparison of features of tools for the design of embedded systems for signal processing.

to make its scheduling decisions; in this sense performing a similar task to that done by ADE, but dynamically, at run-time.

The Matlab [3] system provides an interactive environment for prototyping and testing of system components. One important feature shared by Matlab and ICP is the support for graphical display of signals during debugging sessions. The data-intensive nature of most signal processing applications can make text-based error diagnosis a difficult task. The ability to perform data visualization at all stages of processing can greatly simplify this procedure. Notably lacking from Matlab, though, is support for object-oriented design and the migration of prototypes outside of the interpreted Matlab environment.

Ptolemy [1] represents the current state-of-the-art of design tools for signal processing systems. It fully embraces object-oriented design principles using C++, supports abstract signal representations through object polymorphism in a similar manner to ICP, and offers a graphical metalanguage for system design. This graphical interface, additionally, allows visualization of signal data during system testing. Most notable, however, is that the Ptolemy system was designed to support a wide range of computational and control paradigms. It, thus, differs in scope from ICP, which, by design, provides only one. The difference in code size of Ptolemy (~500K lines of source) and ICP (~12K lines of source) highlights this difference. Further, Ptolemy supports neither the RESUN control paradigm nor the blackboard database model used by ICP, although it could potentially be extended to do so.

6. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of Erkan Dorken and Iftekhar Mahmood, who adapted and implemented the radar clutter analysis application in ICP, and Michael Bosse, who authored the X-Windows interface. Thanks also to Norman Carver and Victor Lesser for consultations on RESUN and IPUS.

7. REFERENCES

[1] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. J. Comp. Sim.*, 4:155–182, April 1994.

[2] M. M. Covell, C. S. Myers, and A. V. Oppenheim. Computer-aided algorithm design and rearrangement. In A. V. Oppenheim and S. H. Nawab, editors, *Symbolic and Knowledge-Based Signal Processing*, pages 30–87. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[3] The MathWorks, Inc., Natick, MA. *Matlab User's Guide*, August 1992.

[4] S. H. Nawab and V. Lesser. Integrated processing and understanding of signals. In A. V. Oppenheim and S. H. Nawab, editors, *Symbolic and Knowledge-Based Signal Processing*, pages 251–285. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[5] CenterLine Software, Inc., Cambridge, MA. *AT&T C++ Language System*, 1990.

[6] N. Carver and V. Lesser. A planner for the control of problem-solving systems. *IEEE Trans. on Sys., Man, and Cybernetics*, 23(6):1519–1536, 1993.

[7] E. E. Milios and S. H. Nawab. Signal abstractions in signal processing software. *IEEE Trans. Acoust. Speech and Signal Processing*, 37(6):913–928, June 1989.

[8] N. Carver and V. Lesser. Blackboard systems for knowledge-based signal processing. In A. V. Oppenheim and S. H. Nawab, editors, *Symbolic and Knowledge-Based Signal Processing*, pages 205–250. Prentice-Hall, Englewood Cliffs, NJ, 1992.

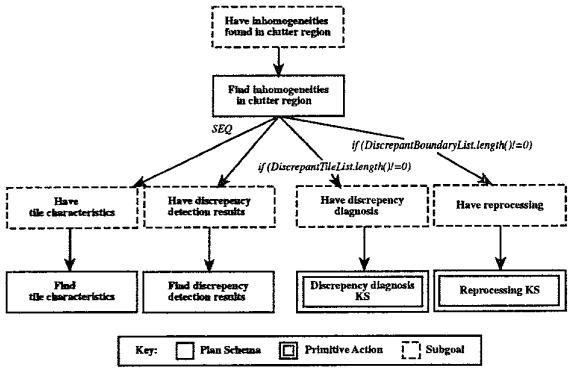
[9] M. Slamani. *A New Approach to Radar Detection Based on the Partitioning and Statistical Characterization of the Surveillance Volume.* Ph.D. dissertation, Syracuse Univ., 1994.

[10] M. Rangaswamy et al. Signal detection in correlated Gaussian and non-Gaussian radar clutter. Technical Report RL-TR-93-79, Rome Laboratory Air Force Material Command, Griffiss Air Force Base, NY, May 1993.

[11] D. Weiner. Private communications, 1991.

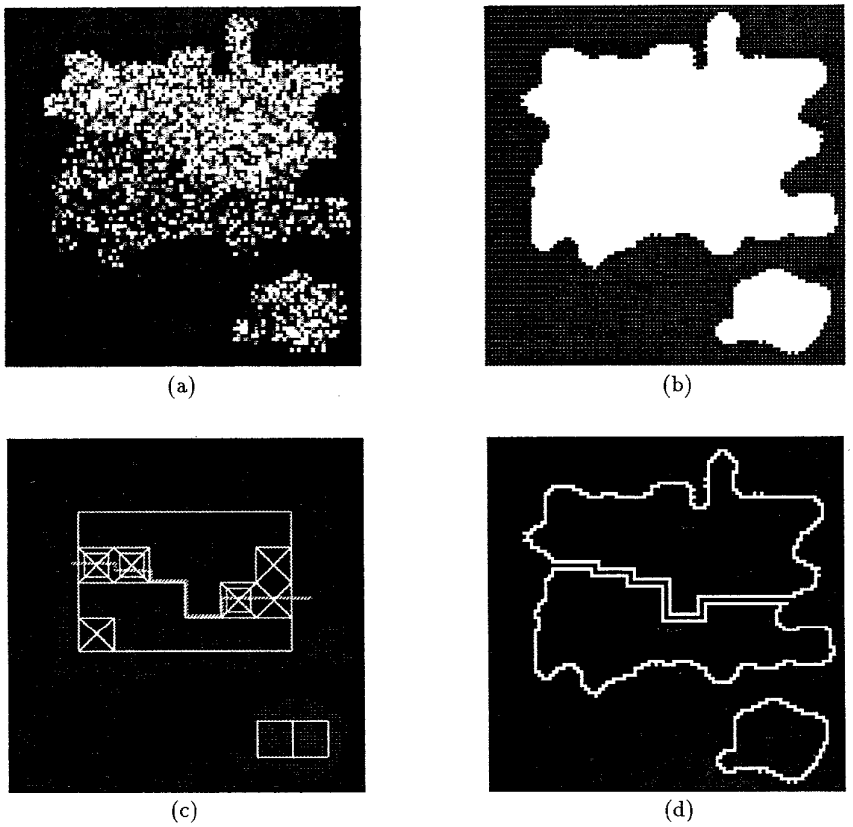
```
DEFINE_NP_PLAN( DiscrepancyDetectionMapLevel,
                PID_DISCREPANCY_DETECTION_MAP_LEVEL,
                SGID_HAVE_DISCREPANCY_DETECTION,
                "SOU *", "int" )

BEGIN_IN_CONSTRAINTS( DiscrepancyDetectionMapLevel )
  if ( inputs.length()) {
    inputs.resetFirst();
    return (((SOU *) inputs.item())->hypothesis->level()
            == MAP_LEVEL);
    )
  else {
    return 0;
  )
END_IN_CONSTRAINTS

BEGIN_SCHEMA( DiscrepancyDetectionMapLevel )
  STEP(1)
    posti( SGID_HAVE_CELLS_RECOVERED, getInput() )
  STEP(2)
    setOutput( (void *) ST_CONTINUE_REPROCESSING_LOOP );
    setStatus( ST_FINISHED );
END_SCHEMA
```

(a)



(b)

Figure 1: (a) Source code for the definition of a short control plan schema using the scripting language. (b) Diagram of a portion of the goal/plan/subgoal hierarchy from a radar clutter analysis application.



(a) (b)

(c) (d)

Figure 2: Graphical output of radar clutter analysis application developed in ICP. (a) Return level (b) Map level (c) Region level (d) Patch level.