Journal of Visual Languages and Computing (1994) **5**, 1–3

Declarative Visual Languages

THERE IS A NATURAL SYNERGISM between declarative programming paradigms and visual programming that is mutually beneficial. Let us consider declarative programming paradigms first. Declarative programming paradigms focus on relationships rather than algorithms. Relationships tend to be multi-dimensional; algorithms tend to be much more single-dimensional. In addition, declarative languages are largely devoid of control flow information; often they are non-deterministic. Thus, they need a representation that does not require, or even imply, an ordering. In summary, declarative languages need an unordered, multi-dimensional representation.

Now, let us consider visual programming. Visual programming offers the potential of improving our ability to communicate and solve problems, provided we use the visual medium to advantage. To do so requires that we realize that visual information is largely unordered and multi-dimensional. This argues that a visual programming environment maximizes its strengths when the underlying paradigm needs an unordered, multi-dimensional representation. Bingo! We have a match.

For these reasons, it is not surprising that declarative programming paradigms and visual programming are increasingly being combined in declarative visual languages. This special issue focuses on some such declarative visual languages and provides examples of some of their diversity of application.

Declarative visual languages emphasize defining relationships existent in the problems they seek to solve and minimizing 'computer programming', i.e. the mapping of these relationships to sequences of operations. It is this 'computer programming' process that is so time-consuming, complex and error-prone. By avoiding it, largely relying upon the underlying system to derive control sequences based strictly upon the relationships and dependencies among the data, we can expedite the programming process, enhance the applicability of the solution and minimize over-constraining the implementation. All three of these benefits are important. Because of them declarative visual languages will continue to grow in importance.

Expediting the programming process is probably the single most important objective of computing science research. The more computer algorithms are constructed by the underlying system, the less programmers must do. Yet, such assistance by the system does not come without a cost. Historically, declarative languages have at times produced unacceptable machine performance. Recent advances in declarative language compilation techniques, along with ever-increasing CPU speed and memory size, have greatly reduced this difficulty. Increasingly, declarative visual languages provide a viable means of reducing programming time.

Enhancing the applicability of the solution is perhaps the next most important objective of computing science research. Again, the more we can leave constructing computer algorithms out of the programming process and up to the underlying system, the more we increase the flexibility of the solution and its future applicability. By restricting the programming process to expressing relationships existent in the problem domain and not allowing these relationships to be intermingled with sequencing constructs, we more readily accommodate future changes. A programmer can change the problem relationships, and the underlying system will derive a new evaluation order.

1045-926X/94/010001 + 03 $08.00/0 © 1994 Academic Press Limited

Finally, minimizing over-constraining the implementation is becoming increasingly important as a goal of computing science research. The fastest and most cost-effective machines in the future will most certainly be parallel. To exploit these machines effectively we will need to consciously remove control and sequencing information from our programs, leaving the machine as much freedom as possible for decisions of evaluation ordering. Declarative visual languages are already largely devoid of unnecessary control and sequencing information.

The four papers in this special issue provide a good breadth of declarative visual language approaches and their applications. Included are four of the more prevalent approaches to declarative visual languages: rule-based, form-based, logic and functional. Of the four papers, two are specific to application areas: program visualization and temporal specification; and two are generic to computer programming.

The first paper, by Kenneth Cox and Gruia-Catalin Roman, describes a rule-based approach to program visualization. The central problem for program visualization is how one associates program events, such as a variable taking on a new value, with an appropriate visualization. Typical approaches have involved a two-step process of, first, instrumenting the program to signal interesting events, and, second, constructing programs that map these interesting events into appropriate visualizations. Visualization systems have provided extensive packages to facilitate primarily the second step, constructing programs that map these interesting events into appropriate visualizations. Cox and Roman provide an alternative that uses a declarative rule-based tuple representation to map program state into visualization state. While the representation used for mapping rules is not a visual representation, it is an excellent example of the use of a declarative language applied to visual objects. After presenting their methodology, the authors examine its computational power and look at its potential use in other areas.

The second paper, by Margaret Burnett and Allen Ambler, discusses visual data abstraction within declarative visual languages. While the abstraction methodologies used in building larger programs have been well-studied for textual procedural languages, they have received considerably less attention in declarative languages and little attention in visual programming languages. Visual data abstraction differs from traditional data abstraction in two ways: (1) it must address the associated graphical representations and interactive behaviors as a part of the definition of abstract data types; and (2) it must support development of visual abstractions entirely through visual programming mechanisms. Burnett and Ambler use Forms/3, a form-based approach to general-purpose programming, to illustrate visual data abstraction. The paper discusses a number of programming-language issues, including maintaining the liveness associated with visual environments, and information hiding and its conflict with visibility in a visual environment. The issue of declarativeness becomes the most pronounced in the consideration of event-handling, which is needed to meet the interactive component of visual data abstraction. An explicit approach to time is used to solve this problem, supporting event-oriented programming while at the same time preserving referential transparency.

The third paper, by L. K. Dillon, G. Kutty, P. M. Melliar-Smith, L. E. Moser and Y. S. Ramakrishna, employs a visual temporal logic approach to stating temporal specifications for concurrent systems. Previous textual approaches to temporal logic specifications, while adequate, have been difficult to read and understand, and have

had little impact on practical applications. Dillon *et al.* use stylized timing diagrams to communicate complex relationships and dependencies. In practice, designers often construct such stylized timing diagrams as a means of understanding and constructing correct temporal specifications. The authors have developed a methodology and a system, GIL (Graphical Interval Logic), for working directly from these diagrams rather than subsequently translating them to some textual form. In addition, while the system designer is constructing specifications, the GIL system provides visual clues to help remind him/her of existing constraints and to discover and correct flaws. The system incorporates a refutation-style proof checker. The paper discusses the design and use of GIL, including developing and proving temporal specifications for the 'fair mutex system' example of concurrent processes.

The fourth, paper, by Jorg Poswig, Guido Vrankar and Claudio Moraga, describes a higher-order functional visual programming language called VisaVis. Programming in VisaVis uses a mechanism termed substitution, in which data and concrete instances of function invocations substitute for more abstract, template-like function invocations. The paper describes the programming process, showing how continuous visual feedback is provided to the user on the status of each function invocation. In addition, it presents an optimized translation of VisaVis programs into FFP, a formal system for functional programming. An important goal of the translation approach was to preserve the data parallelism that is inherent in programs created in VisaVis. The authors conclude with brief descriptions of other research being done using VisaVis. These include animation of visual program execution, work on an implicit-type system and a query interface for browsing the collection of function definitions.

The four papers presented here provide excellent examples of the potential of declarative visual languages. In particular, they explore the merger of declarative programming paradigms and visual programming to achieve systems for expressing problem solutions that expedite the programming process, enhance the applicability of the solution and minimize over-constraining the implementation. There are many other works that might have been presented here as well. In putting together this special issue, we hope that more researchers will take up the pursuit of declarative visual languages for use in these and other problem domains.

MARGARET M. BURNETT AND ALLEN L. AMBLER
GUEST EDITORS