CONF-9607136--1
SAND96-1325C

Visualization and Animation as a Technique to Assist in the Construction of High Assurance Software

Victor L. Winter*
Intelligent Systems and Robotics Center
Sandia National Laboratories
vlwinte@sandia.gov

Abstract

The software construction process consists of a mixture of informal and formal steps. By their very nature, informal steps cannot be formally verified. Empirical evidence suggests that a majority of software errors originate in the informal steps of the software development process. For this reason, when constructing high assurance software, it is essential that a significant effort be made to increase ones confidence (i.e., to validate) that the informal steps have been made correctly. Visualization and animation can be used to provide an "intuitive proof" that the informal steps in the software construction process are correct.

In addition, the formal portion of software construction often permits/demands artistic (informal) decisions to be made (e.g., design decisions). Such decisions often have unexpected/unforseen consequences that are only discovered later in the development process. Visualization and animation techniques can be brought to bear on this aspect of the software construction process by providing a better intuitive understanding of the impact of the informal decisions that are made in program development. This increases the likelyhood that undesirable decisions can be avoided or at least detected earlier in the development process.

1 Motivation

1.1 The Problem

As our society becomes more technologically complex, computers (and the software that they run) are being used in a potentially alarming number of high consequence safety-critical applications. In many safety-critical systems, the reliability of the software component is essential. How can a component of such a system be made more reliable? In engineering, *redundancy* is a standard technique that is used to increase the reliability of a physical component. For example, if an airplane engine breaks down an average of once every 10000 flights, then one can almost be certain that the chance of both the primary engine and a backup engine failing on the same flight will be 1 in 100 million. Such a significant increase in the reliability of a specific component can be achieved through redundancy only if the component possesses a property known as *failure independence.*

*This work was supported in part by the United States Department of Energy under Contract DE-AC04-94AL85000.

Unfortunately, software does not possess the property of failure independence. In fact, software components are fundamentally different from hardware components (e.g., software does not suffer from aging). Empirical evidence indicates that when software fails it is often due to logic errors within the code or because of inaccuracies in the problem specification. In general, for software components, failure independence cannot be achieved even when multiple solutions are developed by different software teams. Further evidence of this observation was demonstrated in an experiment conducted by Nancy Leveson in which she gave the unclassified requirements of a system that uses data from radar to shoot down enemy missiles to 27 programming teams. Most programs functioned correctly in 99 percent of the cases, however all programs tended to fail in the same situations [6]. This experiment demonstrates that redundancy does not guarantee a significant enhancement in software reliability.

A promising solution to the software reliability problem is provided by *formal methods.* Formal methods provide a framework where the correctness of a large portion of the software construction process can be formally proved. The High Integrity Software Project (HIS) at Sandia National Laboratories is developing formal methods and tools for the purpose of enhancing the construction of correct software and systems.

1.2 Formal Methods and Graphical Representations

In the previous section we motivated the necessity for formal methods in the software construction process. Unfortunately it is not possible for the entire software construction process to be formal. The formal construction of software can only be undertaken once a correct mathematical model (i.e., a formal model) of the necessary (relevant) portion of the real world (e.g., the physical components and the requirements) has been constructed. In contrast, determining whether a mathematical model is a faithful representation of a real word object is an informal step in the process. The best one can do to show that a formal model is a faithful representation of the physical system is to validate the model by providing some sort of an incomplete proof of the models correctness. For example, one can extensively test the behavior of the model, one can simulate the model, and one can visualize and animate the model. The objective of each of these validation techniques is to increase ones confidence that the formal model does indeed faithfully represent the physical system.

Visual representations of mathematical models can be provided in a 2-D, 3-D, or VR environment. From a validation standpoint the reason for choosing one representation over another should be motivated solely by the human perception of the model. That is, if the software engineer gets a better understanding of the semantics (behavior) of a model by interacting with the model in a VR environment as opposed to interacting with a 2-D representation, then the VR representation should be used.

A valid question at this point is whether there exists a formal connection between the symbolic description of a problem domain, which is used in the formal reasoning process, and its graphical counterpart. We are currently researching how one can establish such a formal connection. Our long-term goal is to create formal connections between high-level graphical representations and implementation programs and possibly even assembly code. Such connections would greatly facilitate the construction of correct software as well as future software maintenance efforts.

2 Our Work

In the HIS project, we are developing a methodology where high assurance software can be constructed via several mutually complementary approaches: object-oriented stepwise refinements [10], multi-agent strategic approach [5], and synthesis and refinement transformations [8] that are carried out within a domain hierarchy. Our perspective is that most often the domain language for a particular problem is not sufficiently abstract (i.e., it is too complex) to allow effective application of synthesis[1]. This realization is one of the driving forces behind the construction of a domain hierarchy. Domains at a level of abstraction higher than the problem domain can be constructed and represent computational paradigms in which the uninteresting details of the problem have been abstracted away. For example, when constructing software controllers for reactive systems, synchronous computational paradigms, such as those in [2] and [1], provide an elegant abstraction model. Another graphical-based abstraction is provided by the Symbolic Timing Diagram paradigm [3].

In our domain hierarchy there will generally be more than one domain above the problem domain (and also several domains below the problem domain). We are developing a theory whereby consistency between domains in a hierarchy can be formally verified. As we mentioned in the previous paragraph, a primary reason for the construction of a domain hierarchy is that multiple domains greatly ease the burden on synthesis, though this comes at the expense of efficiency. In this framework, refinement transformations are used to recoup the loss of efficiency that results from using synthesis within a domain hierarchy.

2.1 Visualization of a Domain Hierarchy

There is a great deal of flexibility (i.e., artistic choice) available when constructing a domain hierarchy. This flexibility corresponds roughly to the traditional design decisions that are made in top down approaches.

Visualization plays a major role in the construction of a domain hierarchy. It is through visualization and animation that the person constructing the hierarchy can better understand the semantics of a particular domain as well as its relationship with other domains in the hierarchy.

2.2 Animation

Animations provide an excellent means for representing behavioral aspects of objects. In addition, animation can be used to enhance the spacial and relational understanding of static objects.

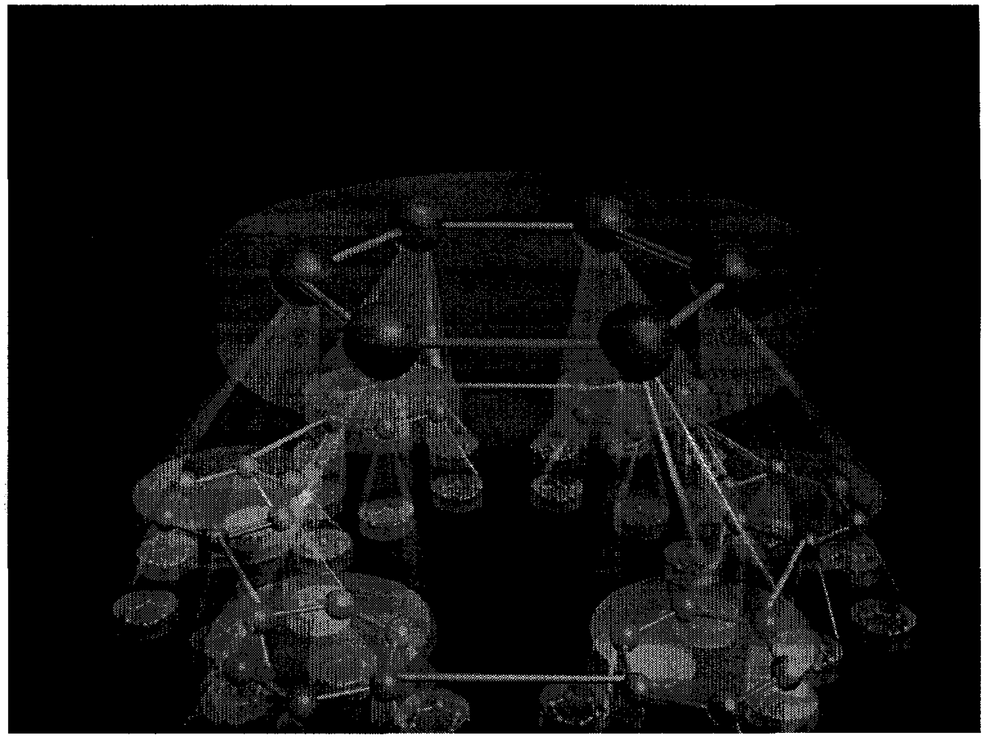
2.2.1 Time

Problem domains, like reactive systems in which *time* is central, can require very complex symbolic descriptions in order to define the behavioral relationships introduced through *time*. In such representations, the intuition behind the formulas can easily be lost. Animation provides a very elegant and natural medium in which the semantics of *time* can be understood.

[1]In our methodology, *synthesis* refers to the use of automated reasoning techniques or sophisticated state space searches such as those found in [5], [7] to create an algorithmic specification from a nonalgorithmic specification.

2.2.2 Complex Static Objects

A domain hierarchy is generally a complex structure. Individual domains possess a certain complexity and connections between domains introduce additional complexity. Such structures are often more naturally represented in 3-D space than in 2-D space. However, visually grasping such a structure by viewing it on a two dimensional screen is somewhat problematic. Animation addresses this problem by providing the observer with the ability to move through the hierarchy. This can greatly increase the intuitive/spacial understanding of the structure. Consider the following figure. An ability to animate the observers perspective provides a tremendous amount of intuitive and spacial understanding.



A Complex Domain Hierarchy

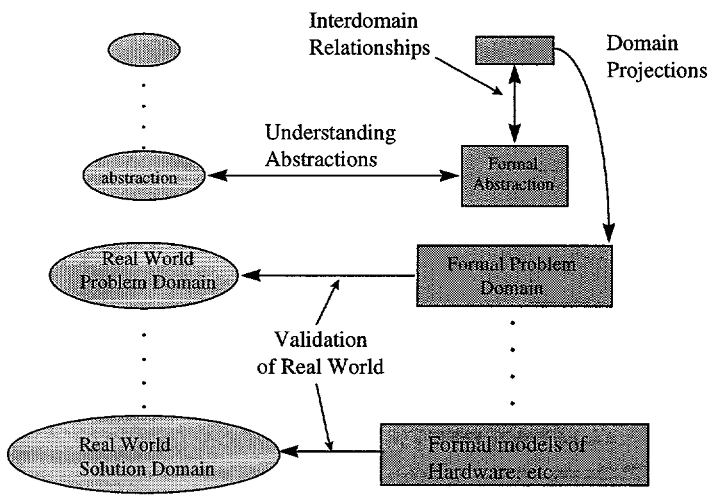
2.2.3 Semantics of State Spaces

The semantics of most domains can naturally be expressed in terms of states and state transitions. For concurrent reactive systems such as the one in [4] concurrent state transitions are possible. Animation of the semantics of such a problem domain together with an interactive interface provide an excellent environment for validating the semantics of the model against the real world system.

In addition, an interactive domain representation provides a means for a user to synthesize correct-by-construction high-level solutions. This realization leads to a synergistic union between mechanical (automated) synthesis and organic (human) synthesis.

2.3 Summary

Imagine two hierarchies, the first contains domains from the real world as well as abstract domains, and the second is a hierarchy of formal domains that is supposed to be a faithful representation of the first hierarchy. The following diagram shows various roles that can be played by visualization and animation to validate the understanding of the hierarchy.


Interdomain Relationships
Domain Projections
Understanding Abstractions
abstraction
Formal Abstraction
Real World Problem Domain
Formal Problem Domain
Validation of Real World
Real World Solution Domain
Formal models of Hardware, etc.

The role of visualization within a domain hierarchy

In the figure above, visualization and animation is used within the software construction process to validate and understand information arising from an informal source (e.g., English text, design decisions, etc.). For example, graphical representations of the formal models of the problem domain can be used to "intuitively validate" that these models are indeed faithful representations of real world objects. Graphical representations can be used to understand formal abstractions, as well as how one level of abstraction relates (projects) into another.

3 Example: A partial domain hierarchy for the production cell.

In our research, we are using the production cell [4] as a general example of a reactive system in order to verify the practical applicability of our research. The production cell is a robotic system consisting of (1) two conveyor belts, (2) one crane, (3) one rotating table, (4) one robot having two arms, and (5) one press. The specification of this problem is given in English. The domain language is at the level of controller commands that turn on/off various motors, electromagnets, and various sensor feedback signals. At this level, motor movements are continuous and system feedback occurs at discrete intervals in response to controller requests. Continuous polling must be performed in order to take the appropriate action in the various system states.

A natural abstraction of this system yields a synchronous paradigm with motor movements producing (instantaneous) discrete state changes. From this domain a further abstraction is possible by removing independent motion from the various components (e.g., a rotating table no longer needs to rotate in order to provide the functionality required of it by the system). This results in a domain where the only events that are expressed are plate exchanges between components (it is up to a lower level abstraction domain to move the appropriate components so that such exchanges can actually occur).

Domains in this hierarchy are essentially state transition diagrams, though the states and the transitions are represented in a manner that conveys more problem specific information than would be displayed in the tradition circle/arrow state transition diagrams. By providing an interactive environment where the behavior of these domains can be explored we have created a natural framework in which the semantics of a domain can be explored and better understood.

We would like to mention that at each step in the construction of the domain hierarchy, it is visualization and animation that provide the intuitive understanding and serve as the catalysts for determining which further abstractions are possible and called for (e.g., how to ease the burden on the synthesis component).

4 Conclusion

Visualization and animation are techniques to assist human understanding of information. In the software construction process there are numerous places where informal information needs to be incorporated within a formal framework. Graphical representations can significantly impact the problems that exist between the informal/formal boundary.

Furthermore, if graphical representations can be formally linked to actual code, then this will have a tremendous impact on software maintenance. In such an environment, a person performing software maintenance can tremendously benifit from much of the effort that has gone into the original development of the implementation.

References

[1] F. Boussinot and R. De Simone. *The ESTEREL Language.* Proceedings of the IEEE, Vol. 79, No. 9, September 1991.

[2] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. *The Synchronous Data Flow Programming Language LUSTRE.* Proceedings of the IEEE, Vol. 79, No. 9, September 1991.

[3] B. Kurshan and J. Katzenelson. *S/R: A language for specifying protocols and other coordinating processes.* In Proc. 5th Annual Int. Phoenix Conf. Comput. Commun., IEEE 1986.

[4] C. Lewerentz and T. Lindner. *Formal Development of Reactive Systems: Case Study Production Cell.* Lecture Notes in Computer Science Vol. 891, Springer-Verlag.

[5] A. Nerode, J. B. Remmel, and A. Yakhnis. *Controllers as Fixed Points of Set-Valued Operators.* Hybrid systems II, Lecture Notes in Computer Science, Vol. 999, pp. 344-358, Springer-Verlag, 1995.

[6] Evan I. Schwartz. *Trust Me, I'm Your Software.* Discover Magazine May 1996, pgs 78-81

[7] B. Stilman. *A Linguistic Approach to Geometric Reasoning.* Int. J. Computers and Mathematics with Applications, 1993, 26(7):29-57.

[8] V. L. Winter and J. M. Boyle. *Proving Refinement Transformations Using Extended Denotational Semantics.* Proceedings of the '96 Durham Transformation Workshop.

[9] V. L. Winter. *Proving the Correctness of Program Transformations.* Ph.D. dissertation, University of New Mexico, 1994.

[10] V. Yakhnis, J. Farrell, and S. Shultz. *Deriving Programs Using Generic Algorithms.* IBM Systems Journal, Vol. 33, No. 1, pp. 158-181, 1994.

5 Biography

Victor L. Winter received his Ph.D. from the University of New Mexico in 1994. His dissertation research focused on proving the correctness of program transformations. Dr. Winter is a member of the High Integrity Software (HIS) group at Sandia National Laboratories. His research interests include trusted software, formal semantic models (graphical-based and symbol-based), theory of computation, automated reasoning and robotics. Dr. Winter can be reached by phone in the United States at (505) 284-2696 or by email at *vlwinte@sandia.gov.*