

SAND REPORT

SAND2003-0095
Unlimited Release
Printed January 2003

Self Organization of Software: LDRD Final Report

Gordon C. Osbourn

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not *infringe privately owned rights*. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.doe.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



SAND2003-0095
Unlimited Release
Printed January 2003

Self Organization of Software: LDRD Final Report

Gordon C. Osbourn
Complex Systems Science
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1423

Abstract

We are currently exploring and developing a new statistical mechanics approach to designing self organizing and self assembling systems that is unique to SNL. The primary application target for this ongoing research is the development of new kinds of nanoscale components and hardware systems. However, a surprising out of the box connection to software development is emerging from this effort. With some amount of modification, the collective behavior physics ideas for enabling simple hardware components to self organize may also provide design methods for a new class of software modules. Large numbers of these relatively small software components, if designed correctly, would be able to self assemble into a variety of much larger and more complex software systems. This self organization process would be steered to yield desired sets of system properties. If successful, this would provide a radical (disruptive technology) path to developing complex, high reliability software unlike any known today. The special work needed to evaluate this high risk, high payoff opportunity does not fit well into existing SNL funding categories, as it is well outside of the mainstreams of both conventional software development practices and the nanoscience research area that spawned it. We proposed a small LDRD effort aimed at appropriately generalizing these collective behavior physics concepts and testing their feasibility for achieving the self organization of large software systems. Our favorable results motivate an expanded effort to fully develop self-organizing software as a new technology.

Background

The development of complex, low-defect software packages is a difficult and expensive task. A 1995 survey indicated: 5 out of 6 corporate software projects don't produce code that performs as required; 1 out of 3 corporate software projects are canceled (primarily when development costs and development times escalate beyond affordable limits). The cost of complex software development has several components. Highly trained software engineers are expensive. Software development requires the *perfectly correct* specification of vast amounts of detail. This makes software development very time consuming and error-prone. Discovering errors and eliminating them correctly has also become a major expense. The combinatorial explosion of possible inputs and decision paths means that complex software can never be fully tested for flaws. Further, it has been estimated that 1/5 to 1/3 of software changes that fix a known bug inadvertently introduce a new (but now undiscovered) bug. Even the most "trivial" changes in an apparently working piece of complex software create a large chance of breaking the program in some "invisible" way. Errors are never fully eliminated from today's large commercial software packages. The growing dependence of our society and our military on COTS software makes these cost issues and defect issues ones of increasing national concern.

Computer scientists have long sought an approach to software development that would solve the above problems. Many useful techniques for improvement, including structured design, software patterns, code reuse and object oriented design have been championed in the last two decades. All of these approaches have provided definite benefits. However, none of these has been the desired "silver bullet", i.e. none have reduced the development costs by orders of magnitude. A key problem in software development is that there is no room for uncertainty in the software commands. Every minute detail of every task must be explicitly, completely and perfectly described somewhere in the software code. Any and all special cases that arise must be explicitly addressed. All possible situations, user input combinations, ranges of variables and so on should be anticipated in the design to avoid "crashes". These requirements are essentially

impossible to meet while developing complex software codes using existing techniques. Further, they tend to clash with the way that human minds deal with complex tasks.

Some computer scientists have dreamed of a general-purpose "4th generation" software language or a highly advanced software development tool that would require only the specification of software requirements. The actual detailed code to meet the requirements would be generated by the advanced tool, rather than by software engineers. This would save time directly by avoiding many of the man-years now spent in detailed code design and generation. Many of the programming errors generated through common human mistakes would be eliminated as well. This would further save considerable time/expense and improve the software reliability. No one to date has discovered a method to automatically generate large and complex software programs in this fashion.

Much research has been carried out on genetic programming techniques with this same long-term goal. Genetic programming was motivated by concepts from outside of the software field, i.e. by Darwinian evolution. Populations of programs are randomly constructed and randomly modified. The desired actions of the software must be expressed in terms of a fitness function. The execution of each program is evaluated against this fitness function, and the best programs are retained as the next generation. This approach has generated interesting programs when evolved on massively parallel computers for sufficiently long times. The resulting software can exhibit novel structure that a human software engineer would not consider (or necessarily even understand). At present, scalability to software applications with large-scale requirements, the accumulation of "introns" (significant chunks of the code that do nothing useful), and the use of fitness functions to specify software actions are key hurdles for considering genetic programming for general purpose software development.

The purpose of this small (sub-FTE), short-term (< 1 year), "out-of-the-box" project was to carry out an initial investigation into the feasibility of adapting physical science notions of self-assembly and self-organization to this difficult software problem. Given the background discussion above, it should be clear that this project is very high risk. We have initiated a new research effort to understand physical and biological hierarchical self-organization processes. Hierarchical self-organization is the process by

which biological systems build themselves through separate self-organization processes at different length scales. Understanding this process is a major scientific goal in the field of complex adaptive systems. The application goal is to develop new self-organization assembly methods for creating large-scale physical machines, electronics and systems out of nanoscale and microscale components. We are developing a non-equilibrium statistical mechanics model for understanding the collective behaviors that underlie a variety of physical self-organization processes at different length scales. The key idea is that self-organization can be generally viewed as an entropy-reducing process, and so should be achievable with appropriate sets of components that can drive themselves to far-from-equilibrium collective states. The self-organized collective states can then act as larger-scale components that, under specially designed circumstances, can carry out their own self-organization process to create even larger scale hierarchical collective states. This physics viewpoint provides novel insights into the properties that physical systems must possess to self-organize.

We hypothesized that, with some modifications, the process of entropy-reducing hierarchical self-organization might be applied to the self-assembly of software systems. The long-term goal is to enable the limiting of human input to specifications of the desired software tasks, rather than descriptions of how to carry out these tasks. These specifications would directly constrain and guide the software self-organization process towards the desired functionality. It is also expected that the general format for specifying the desired actions of the self-assembling software would provide an intuitive user interface. The software modules self-assembled in this fashion would themselves retain the capability of combining with each other to further self-organize into larger, hierarchical software packages. An interesting additional property of the self-assembled software modules will be that they can adapt (or reorganize) when presented with a change in the original specifications. This property greatly expands the possibilities and reliability of both modifying and reusing such software.

No existing software development approach has been based on physics concepts. The relevance of non-equilibrium statistical mechanics concepts to software development is a new and non-obvious hypothesis that is very far outside-of-the-box. This LDRD

proposal enabled some small-scale tests of the feasibility of applying our statistical mechanics concepts to achieve the self-organization of software modules. Our purpose was to determine if a larger-scale follow-on effort should be launched to fully develop this new approach.

Project Activities

Translating physical self-assembly designs to software self-assembly designs

The heart of our effort has been an ongoing effort to develop a theoretical understanding of the non-equilibrium physics of self-organizing processes across multiple length scales. This project has drawn heavily on this parallel research effort. However, there are clearly fundamental differences between a system of software instructions in a computer system and an actual physical self-assembling system. It was necessary for us to explore how the key physics concepts can be best translated to the software arena. A listing of these translation issues includes (i) converting from three-dimensional physical space to the essentially one-dimensional address space of computer memory (ii) providing a surrogate for physical energy (iii) converting from parallel physical activity to sequential processor execution (iv) providing surrogates for physical constraints and physical forces (v) providing surrogates for physical fluctuations and stochastic events (vi) providing surrogates for chemical bonding. We also considered the potential associated with properties of software systems that are not present in physical systems. Examples of these properties include moving arbitrary “distances” in an execution time step and the ability to copy any software structure directly.

The exact details of how these translations can best be made will be the subject of follow on development of our approach. Here we note that a useful idea for representing energy in our software system is to associate it with time. For dynamic processes, this means the amount and priority of microprocessor execution time allocated to that process. Processes that are driven by larger amounts of energy proceed more rapidly by having greater and prompter access to execution time. For potential energy barriers, e.g. in maintaining chemical bonds, this means the lifetimes of the software entities that represent physical energy storage.

Self-assembly processes

Providing surrogates to bonding is essential for mapping physical assembly to the assembly of software instructions. There are two types of bonding connections between software components that can be represented readily in software. One is to use pointers to provide direction from one component to the next. The other is creating proximity of the components in the memory space. Pointers can be redirected with a single memory overwrite, and so are useful for bonds that are to be created and broken frequently. However, the execution time of software built with such bonds is slowed by the additional indirection associated with the pointers. Proximity-based bonds are created by moving the complete components to a free location in memory so that the components are at contiguous locations. This process is much slower than reassigning pointers, and is appropriate for stable bonds that have long lifetimes. The corresponding execution times for software structures built with these bonds are generally less than those using pointer bonds. Given the use of processor execution time as an energy surrogate, we can see that it takes more energy to create a proximity bond than to create a pointer bond.

Again, the full details of our self-assembly methodology will be the subject of follow-on work. For this feasibility study, we were able to successfully self-assemble some very simple components using the bonding types described above, execute them, and then disassemble them. This was all done without recompiling using a Forth-based infrastructure outlined below.

Developing and testing an infrastructure for "self-modifying" software

A key hurdle in developing self-organizing software is that conventional software languages (e.g. C/C++) have both a fixed, complex syntax and a complex, "offline" compilation process. These attributes are not designed for, and are essentially inconsistent with, the need to have efficient, real-time self-assembly and self-modification of software instructions carried out by the software itself. Our past experience with the Forth language suggested that it is particularly well suited for our purposes. Forth effectively has no syntax, so that software instructions can be assembled without concern for syntax structures or keyword use restrictions. Forth also has a direct,

efficient and “real-time” technique for compiling software instructions into memory. This allows the software to directly modify itself while running without the offline compilation step that would be required by many popular software languages. Finally, modern Forth implementations for Pentium computers can execute programs with a speed comparable to popular languages with highly optimized compilers (e.g. C/C++). A significant part of our effort was in exploring the use of Forth for developing the non-standard infrastructure needed to enable software self-modification. This work confirmed that Forth provides a good foundation on which to build a self-organizing software technology.

Software that adapts to new goals

Our approach to this is to mimic the behavior of non-equilibrium biological systems. These systems constantly renew themselves by the programmed death of existing system components and replacement of outgoing components with new system components. Further, under changing conditions, the biological systems can generate different sets of replacement components, so that the system components and their functionalities both change along with changing conditions. Software adaptation requires these two properties -- self-renewal ability along with the ability to generate and “steer” distinct replacement components that meet new requirements. We will be exploring the second capability in a follow-on effort. This project experimented with self-renewal and programmed death of components. Much of the work here was focused on simulations of physical systems in which the components actually have a finite useful lifetime, because the process is more difficult in physical systems. Using the physics simulation results (that will be described in detail in another publication), we can see how to design self-renewal processes in our software systems so that the functionality can be modified as requirements change. These adaptive changes can occur while the software is still running. This was a key milestone and provided considerable support for the overall feasibility of our approach.

Project Conclusions and Future Directions

Despite the small funding level and short time scale of this exploratory project, we were able to make sufficient progress to provide strong support for our hypothesis that physical self-assembly and self-organization processes can be translated into an analogous methodology for self-assembling software. Thus, we were both able to meet our milestones and provide justification for an expanded effort to fully develop self-organizing software as a new technology. This expanded effort is now underway (beginning in FY03).

We will be carrying out much more work on the steering mechanisms to guide the software adaptation. We also need to work on the hierarchical aspects of software self-organization. This is a key requirement for scaling up this technique to large software systems. Again, we will follow the lead provided by our physical self-organization research to achieve automated hierarchical structuring of these software systems as they increase in size.

Given the indications (provided by this project) that self-organizing software is indeed feasible, we note that there are strong motivations for aggressively pursuing this new approach. Successful development of this approach would produce a "disruptive" technology for cost-effective software development. "Disruptive" technologies (e.g., transistors/ICs vs. vacuum tubes) are often too radically different to be accepted by the existing market leaders. We can expect that large software development organizations with firmly entrenched software development methodologies, either inside or external to SNL, will NOT be the early adopters or early supporters of this radical technology. In the marketplace, newly formed businesses may use disruptive IP to eventually overrun the markets of the original technology. The current software market is enormous, and is likely to experience continued growth. A new method for developing reliable, complex software codes at greatly reduced costs would ultimately broadly impact the U.S. economy. The development of large software products for National security and critical infrastructure applications might provide surety at greatly reduced costs. As we develop a prototype of this software development capability, we hope to test and validate it by

developing useful National security software tools for carefully selected internal SNL partners.

SNL generation and ownership of IP for such a broadly applicable, "disruptive" technology might prove highly beneficial in the long term. New understanding of how complex self-organization processes can be guided towards useful behavior is of considerable scientific interest as well. We expect to encounter challenges in the conflicting goals of rapid and complete disclosure of our multitude of physical and software self-organization ideas to the scientific community (before they have been fully implemented or exhaustively tested and explored) versus thorough validation and protection of all aspects of the associated IP to best enable future licensing opportunities.

Acknowledgements

This work was sponsored by the U.S. Department of Energy under Contract DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corp., a Lockheed Martin Co., for the U.S. Department of Energy.

Distribution:

1	MS-0188	LDRD office, 4001
10	MS-1423	G.C. Osbourn, 1001
1	MS-0612	Review & Approval Desk, 9612, for DOE/OSTI
2	MS-0899	Technical Library, 9616
1	MS-9018	Central Technical Files, 8945-1