

Dr. John L. Kulp, Jr. / MIT

MODIFICATION NOTICE

The following report has been modified from its original version as follows:

Text formatting has been changed

by the addition of paragraph breaks, page breaks and enlargement of the type face.

INTELLECTUAL PROPERTY NOTICE

The original text was written by Dr. John L. Kulp, Jr. The notices in this document do not intend to claim any rights to the part of this work that is intellectual property of Dr. John L. Kulp, Jr.

completely novel software technology
SUMMARY
INITIAL IMPRESSIONS
BRIAN'S PRESENTATION
THE TECHNOLOGY
ASSESSMENT
summary

Dr. John L. Kulp, Jr.
5 Hers Court
Tilleville, NJ 08500
(609) 730-8372

Technology Report:
Authorgenics, Inc.

.....
A start-up company with a completely novel software technology for the automatic construction of applications from high-level specifications.

SUMMARY

Brian Stack, founder of Authorgenics, is a brilliant fellow who has developed a completely novel approach to generating applications from high-level specifications. Because it is based on new principles and concepts, the means by which this software accomplishes its function is very complex to explain. It cannot be related to previous approaches to application generation or CASE tools. The software has demonstrated remarkable performance in the hands of its creator and his associates. It has been applied to database and transaction style applications (or those that can be cast in those terms). It is not refined enough yet for use by a broader audience (hence the current investment offering), but such enhancements do not appear to present an undue challenge.

As with all revolutionary products, market acceptance of a new paradigm involves significant uncertainty, and depends on some combination of evangelism, positioning, partnering, timing, and product adaptation. Excellent support for the new corporate intranet/extranet environment is an example of a market window where many companies (or their IS suppliers) will be writing new applications to capitalize on the new context. Authorgenics' technology could be the basis for a major success in the software industry. Whether this success is realized will depend on the ability of the CEO (to be hired) to execute a strategy which addresses these issues.

INITIAL IMPRESSIONS

When I arrived in Miami, I was pretty skeptical. True revolutionary changes in the industry are extremely rare. I experience a significant cognitive dissonance when presented with computational

analogies with the human brain which are inevitably shallow given the complexities involved. I was a cofounder in 1980 of a company that made claims of this sort (Symbolics).

Further, I wanted to understand the scope of applicability for this technology (more on this later), but was told that it was probably unachievable. We were warned that Authorgenics' technology was so different that we would not be able to relate it to existing software paradigms (this turned out to be largely true). So, I found a lot of the opening statements fairly implausible, but was intrigued and curious to know how this product was able to produce the code generated.

BRIAN'S PRESENTATION

Brian opened the day with a discussion on the epistemology of the words: describe, discover, invent and create. The point was that he views deep similarities in the cognitive process behind these concepts. Thus, describing an application, and creating that application (given a "perception" of its description) can be based on common machinery. Pretty abstract, but it sets a frame of mind and vantage point for thinking about intelligent processes.

Next he surveyed the existing approaches to intelligent systems: rule based expert systems, neural networks, symbolic pattern matching, genetic algorithms, and fuzzy logic. All have some properties that are desirable for achieving intelligent functioning of software, but all are limited in one way or another. He needed to find a way to subsume the best attributes of all of them.

Then he went into a long discussion of the nature of perception. The central idea being that all stimuli are interpreted or represented in terms of previously experienced stimuli (the analogy for application specification and application generation is that specifications are represented in terms of previously encountered specifications, and code is generated in terms of previously generated code). None of this says yet how things are structured or organized (that came later) which is the root of the issue.

About this time we broke for lunch. We still had no idea how Brian's software worked, but I was certainly convinced he was a very smart fellow with a broad grasp of a lot of issues in cognitive processing, both artificial and natural. The morning discussion was stimulating, but frustration was building that we weren't getting to the central issue of how the software actually works.

THE TECHNOLOGY

In the afternoon, he gave us an overview of the software from a PowerPoint presentation. This included high-level, abstract diagrams showing a variety of related concepts. The terminology used was unfamiliar and not explained too well, and didn't mean too much without good examples. What I got out of it was that there are two repositories of information on which the system is based. The system "works" as a side-effect of the pattern resolution mechanism that works between these repositories.

One, called the Genetic Database, contained information about how programs are created, including permanent, application-independent "rules," as well as information about the current application being developed. It contains layers of semantics corresponding (I suspect) to conceptual levels of programs

(program structure, style, syntax, skills). How this is actually represented isn't clear other than that it is not explicit but the some form of implicit encoding which is the result of some unspecified training or construction process. This was the source of much obscurity and confusion in that if you ask "Where is the rule represented?" the answer is "Nowhere in particular" because (I presume) the rule has been internalized as some pattern of references in the database.

The second database called the Molecular Database contained information for describing applications. It too was layered with a variety of concepts (although I can't recall them off hand).

We saw one example of actually generating a simple application. The specification involved some high level statements (e.g. select file) and more detailed statements involving selecting types of fields for a data record. The latter operation was accomplished by selecting items from menus. Brian called this specifying feature packets, and that this level of specification would not be done directly by the user in the future. My (vague) understanding of what happens when you select one of the menu items is that the item is interpreted in terms of the current contents of the database.

This would be easier to understand if there was a very simple "toy" example to show how the database works. Namely, show the definition of a trivial "language" and a one or two step "application" and show how they are represented (or in-terms-of what). Perhaps even this is not possible if there is a certain critical mass needed to do anything, and understanding that critical mass in detail is a huge undertaking.

One remarkable fact is that this same machinery can generate natural language text as well as computer language.

In examining the code generated by Author, I was on one hand amazed that the web of implicit encodings should actually produce organized, well-formatted code. On the other hand, the code wasn't particularly optimal. The particular thing I noticed was redundant recomputing of condition predicates. This leads me to wonder what sorts of global optimizations or common subexpression abstraction can fall out of the process.

We had a discussion about the need to support object-oriented programs. Brian made a point that in Author, the whole program can be "seen" at once, thus negating the need for the kinds of modularity provided by object-orientation. I didn't agree with this (or rather, only agree that this is true for standalone programs of modest size and complexity). Brian didn't seem to have much experience writing large object-oriented programs, so his understanding was mostly theoretical. We didn't get to argue it out, so maybe I'm missing some point of his, but in my experience, the kind of modularity provided by object-oriented construction provides intrinsic efficiencies in the information structure of programs. Saying that you can rebuild the whole system from scratch every time you want to add some functionality is impractical, no matter how automated it is.

ASSESSMENT

I draw conclusions with some trepidation due to the limited exposure I had to how things actually work, and thus, my limited resulting understanding. The Authorgenics technology that Brian has created is clearly an impressive thing. Because it is difficult to understand, it is hard to know if its current limitations are fundamental or easily overcome.

The current user interface is very dated. This is an almost trivial point and can probably be easily overcome. The program is implemented in Business Basic. To someone experienced in implementing complex software of millions of lines of highly integrated, object-oriented code in C++, Java, or Common Lisp, the program composition limitations of Basic are evident. My guess is that this turns out not to be too important for Author, since, if I understand it, the program logic of Author is not that complex. The complexity is in the data! This is one of the striking features of this program. Having said this, being written in Basic leaves an aura of being unsophisticated from a software engineering perspective. The same would very likely execute much faster if written in a more efficient language.

A key issue for this technology is how well it works in the context of existing API's (application program interfaces). In the emerging multi-vendor standards-based environment, almost all applications will not be stand-alone, but will interface to a myriad of other systems using a variety of protocols. Author clearly can talk to multiple databases (how? SQL?) and generate character-based reports and screens. It's not clear how it generalizes to systems built out of component software (OLE, ActiveX, Java Beans, etc.), directory protocols (LDAP), COBRA interfaces, messaging interfaces, dynamic/interactive GUI screen layout, drag-and-drop functionality, typographical-elaborate reports, etc.

The ability to support other programming languages and paradigms has not really been demonstrated. By "support," I mean more than "generate code in". I can readily take a Basic program and rewrite it in C++ without using any of the distinctive features of C++. By "support", I would mean construct class abstractions, interface classes, protocol classes and the like from abstract application specifications. This strikes me as really hard and I'd be even more impressed if it could be done. I don't think it can be.

Another question is how broad a class of applications can be supported. It is claimed, for instance, that process control applications have been supported. Well, not really. There are some basic process control requirements that can be satisfied by modeling sequence stated by transactions. What most people mean by process control are applications requiring execution under real-time constraints, complex event scheduling, resource planning, adaptive control demonstrated, to my knowledge. Having said this, it's not obvious that Author cannot be extended to address any new computation idioms, but some may be very hard.

Another question is how successfully can a new layer be implemented that eliminates the need to directly specify "feature packets". Although this seems possible it is not obvious. Without this new layer, a potential user of Author would have to become proficient in the semantics of these times to successfully use the system for real applications. There is some evidence that Authorgenics employees have accomplished this, so it can be done. But it might be issue for outside customers.

A last concern I have is whether anyone but Brian can understand the internals of Author well enough to be able to extend it to new languages, new interfaces, or new application semantics. As I understand it, such extension involves "training" the system with appropriate examples. I have no idea how hard or easy this is, but given the difficulty I had in understanding the system, I can't imagine it's easy. It is essential that he build a team around him that has a full grasp of these issues.

In summary, the Authorgenics technology is completely novel and substantive. It has demonstrated a remarkable capability to generate database/transaction style applications in Business Basic, interface to multiple databases, and create character-based user interfaces. Equally remarkable is its ability to

generate reference documentation that is accurate and highly readable. Questions remain as to whether this technology is very difficult to understand is both a strength and a weakness. A strength, because it will be very difficult to copy, therefore easy to protect. A weakness, in that it may scare potential customers off, and it may be difficult to expand its core development and evolution beyond a single individual.