NanoDesign: Concepts and Software for a Nanotechnology Based on Functionalized Fullerenes

Authors

Al Globus, MRJ, Inc. at NASA Ames Research Center, and Richard Jaffe, NASA Ames Research Center.

Abstract

Eric Drexler [Drexler 92a] has proposed a hypothetical nanotechnology based on diamond and investigated the properties of such molecular systems. While attractive, diamonoid nanotechnology is not physically accessible with straightforward extensions of current laboratory techniques. We propose a nanotechnology based on functionalized fullerenes and investigate carbon nanotube based gears with teeth added via a benzyne reaction known to occur with C60 [Hoke 92]. The gears are single-walled carbon nanotubes with appended o-benzyne groups for teeth. Fullerenes are in widespread laboratory use and can be functionalized in many ways [Diederich 96]. Companion papers computationally demonstrate the properties of these gears (they appear to work) [Han 96] and the accessibility of the benzyne/nanotube reaction [Jaffe 96a]. This paper describes the molecular design techniques and rationale as well as the software that implements these design techniques. The software is a set of persistent C++ [Stroustrup 91] objects controlled by TCL [Ousterhout 94] command scripts. The c++/tcl interface is automatically generated by a software system called tcl_c++ developed by the author and described here. The objects keep track of different portions of the molecular machinery to allow different simulation techniques and boundary conditions to be applied as appropriate. This capability has been required to demonstrate (computationally) our gear's feasibility [Han 96]. A new distributed software architecture featuring a WWW universal client, CORBA distributed objects, and agent software is under consideration. The software architecture is intended to eventually enable a widely disbursed group to develop complex simulated molecular machines.

To the full paper.

References

To companion papers.

NanoDesign: Concepts and Software for a Nanotechnology Based on Functionalized Fullerenes

Al Globus, MRJ, Inc. at NASA Ames Research Center and Richard Jaffe, NASA Ames Research Center.

Organization

- Abstract
- Introduction
- Fullerene Nanotechnology
- Design Software
  - Current Software Architecture
  - Proposed Future Software Architecture
- Conclusions

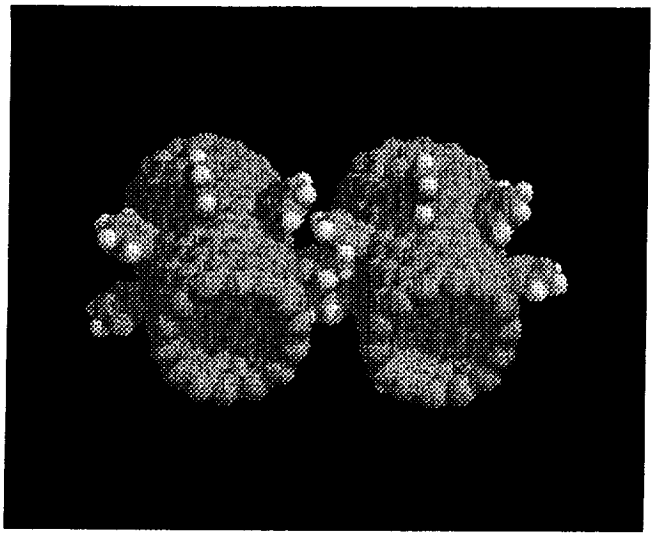
Introduction

Eric Drexler [Drexler 92a] has proposed a hypothetical nanotechnology based on diamond and there is informed speculation that this technology could have tremendous aerospace applications [Globus 96b]. Unfortunately, no one knows how to build diamonoid components in the laboratory. To gain the benefits of nanotechnology, a more accessible chemical basis is needed. We have chosen to investigate fullerene nanotechnology and develop software to support this work. Software development is at a very early stage. This paper is a status report, not an exposition of finished work.

Fullerene Nanotechnology

A nanotechnology based on fullerenes has been suggested by others. C60 and other cage-like fullerenes provide points, carbon nanotubes provide lines, and these can -- in principle -- be combined to create three dimensional objects. Since fullerenes can be functionalized by a wide variety of molecular fragments [Dresselhaus 96], a wide array of objects with many properties may be created. One measure of the accessibility of fullerenes is the number of patents that have been issued. Another is this email advertisement I received in September 1996 selling fullerenes by the gram.

The first systems we have investigated are various gears built out of single walled carbon nanotubes with o-benzyne groups attached to form the teeth. [Thess 96] has demonstrated a 70% yield in carbon nanotube production so the tube should be synthetically accessible, although [Thess 96] generated (10,10) tubes whereas most of our simulations use (14,0) tubes. [Hoke 92] has shown that benzyne reacts with C60 to form a 1-2 bond between six membered rings and quantum calculations [Jaffe 96a] suggest that a similar reaction should take place on carbon nanotubes, although 1-4 bonds are slightly preferred. Adding aromatic rings to the tube should give us relatively stiff molecular gear teeth, and this has proved to be the case [Han 96].



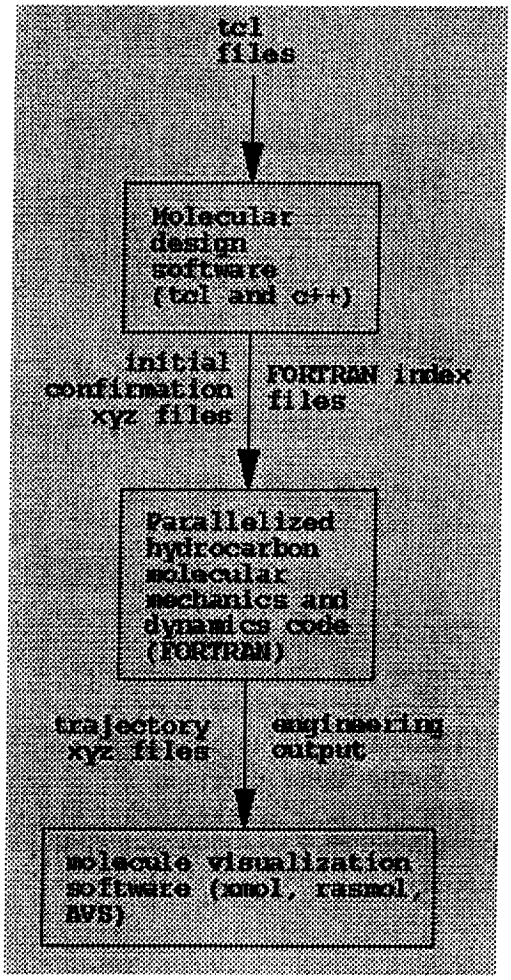
A typical gear configuration.

Using the NanoDesign design and simulation software described below, [Han 96] has shown that -- assuming you believe the force field -- a number of gear and gear/shaft systems will function mechanically in a vacuum. These simulations used a software thermostat and motor, but there is reason to believe that physical implications of these functions can be provided. Preliminary simulations suggest that cooling is possible using an inert atmosphere. Experimental evidence (Sunny Bains reports in *Science* , volume 273, 5 July 1996, p. 36 on upcoming papers) and simulation [Tuzun 95] suggest that lasers may be used to rotate the gears. The tube is functionalizing with positive and negative charges in appropriate locations and the lasers are used to create a rotating electric field.

Design Software

The simple molecular machines simulated so far can be easily designed and modeled using ad hoc software and molecule development. However, to design complex systems such as the molecular assembler/replicators envisioned by the NASA Ames Computational Molecular Nanotechnology Project [Globus 96b], a more sophisticated software architecture will be needed. The current NanoDesign software architecture is a set of c++ classes with a tcl front end for interactive molecular gear design. Simulation is via a parallelized FORTRAN program which reads files produced by the design system. We envision a future architecture centered around an object oriented database of molecular machine components and systems with distributed access via CORBA from a user interface based on a WWW universal client.

Current Software Architecture



Current NanoDesign software architecture.

The current system consists of a parallelized FORTRAN program to simulate hydrocarbon systems. Supramolecular conformations come from xyz files (the force field does not require a bond network in the input) produced by a c++ and tcl program using the tcl_c++ interface generator. The software also creates FORTRAN files with indices into an array of atoms indicating where each component (e.g., gear teeth) begins and ends. The user creates tcl files with tcl functions to create and modify c++ objects. For example, this tcl fragment creates a buckytube:

```
# create a buckytube
set tube [aBuckytube]
# it will be 14,0 tube
$tube setRingCircumference 14
# make it 21 rings long
$tube setRingLength 21
# set the FORTRAN variable name for the tube
$tube setVariableName "tube"
# tell c++ to create the tube
$tube build

# write the confirmation into a file
$tube writeXyz "tube.xyz"
# write the FORTRAN declarations and index assignments into a file
$tube writeFORTRANVariables "tube.f"
```

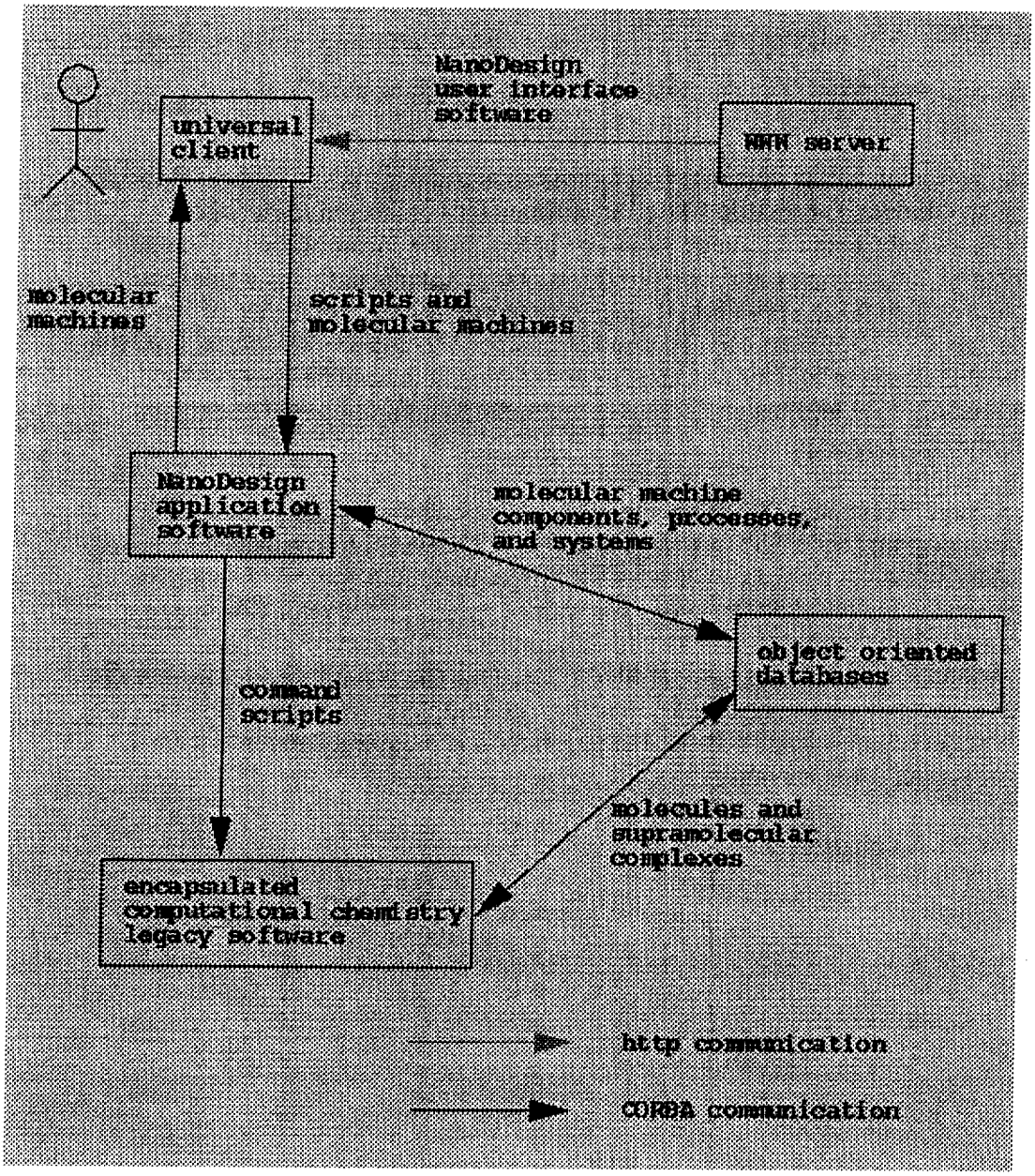
See here for details on the FORTRAN output.

tcl_c++

C++ was chosen for molecular design for its object oriented properties and high performance. However, c++ is a compiled language so changes to the code take a bit of time. This is inconvenient when designing molecular systems; an interpreted language would be better. Tcl is meant to be used as an embedded interpreted command language in c and c++ programs. Tcl [Ousterhout 94] is a full featured language with loops, procedures, variables, conditionals, expressions and other capabilities of procedural computer languages. C++ programs can add new tcl functions to any tcl interpreter linked in. Thus, tcl gives us an interpreted interface to the c++ class library so molecules can be designed at interactive rates. Note that both Cerius2 and Insight/Discover commercial computational chemistry packages use tcl for their command language.

The Visualization Toolkit project [Schroeder 96] discovered that a tcl interface to a large c++ class library can require substantial programmer effort to write the glue that allows tcl to control c++ classes. The vtk project avoided this by writing a partial c++ header file parser that reads the c++ header file for a class and automatically generates the tcl interface code. We wanted more control over which c++ member functions were tcl accessible, so the tcl_c++ system requires a file for each c++ class to define which member functions, variables, and constants are tcl accessible. This file is read by a tcl interpreter with tcl procedures defined to generate c++ code to allow another tcl interpreter to control the c++ class in question. Fortunately, although tcl_c++ itself was hard to program, it is easy and convenient for a programmer to use. For details of tcl_c++ see here.

Proposed Future Software Architecture



Future distributed NanoDesign software architecture. Note that each box may represent many instances distributed onto almost any machine.

The current NanoDesign molecular design software appears to the user as an interpreted language based on tcl. This is very effective for design of simple parts and systems. To design and computationally test complex replicators will require a more sophisticated system similar to the mechanical CAD systems available in the commercial marketplace. Furthermore, it would be of substantial practical advantage if the design team could be geographically dispersed. Therefore, we are investigating an software architecture based on a universal client (for example, a WWW browser), CORBA distributed objects, an object oriented database, and encapsulated computational chemistry legacy software. We are also interested in using command language fragments to control remote objects. Software that communicates this way is sometimes called agents [Genesereth 94].

Universal Client

With the advent of modern WWW browsers implementing languages such as Java and JavaScript, it is possible to write applications using these browsers as the user interface. This saves development time since most user interface functionality comes free, integration with the WWW is trivial, and the better browsers run on a wide variety of platforms so portability is almost free. VRML can be used for 3D graphics and plug-ins such as the recently announced Biosym/MSI, Inc. molecule browser provide crucial functionality without much work.

Recently, Netscape, Inc. announced that the netscape WWW browser would be made CORBA (see below) compliant offering a standard way to communicate between application code loaded by the browser and databases and computational chemistry software resident on servers and supercomputers. Previously, only the stateless http protocol was available to web browsers. Hopefully, other companies in the extremely competitive WWW browser market will follow suit.

These developments suggest that a single program can function as the user interface for a wide variety of applications, including computational nanotechnology. These applications load software (e.g. Java applets and JavaScript) into the browser when the user requests it. The applications then communicate with databases and remote objects (such as encapsulated legacy software) to meet user needs.

CORBA (Common Object Request Broker Architecture)

The universal browser is of little use in developing complex molecular machines if it cannot communicate with databases of components and systems and invoke high performance codes on fast machines to do the analysis. CORBA, a distributed object standard developed by the OMG (Object Management Group), provides a means for distributed objects -- for example the universal browser application, a database containing an evolving molecular machine design, and simulation codes -- to communicate and control each other. The simplest description of CORBA is that each object is represented by an interface described by the CORBA IDL (interface description language). Operations and data defined in the IDL may be accessed by other CORBA objects on the network. System software (called ORBs -- object request brokers) is responsible for communicating between objects whether they be on the same machine or widely distributed. See [Siegel 96] for a description of CORBA.

Object Oriented Database

To develop complex molecular machines, databases of components and processes as well as complex databases describing individual systems will be required. Object oriented databases appear to be better than relational databases for design systems for products such as aircraft and molecular machines.

Encapsulated Computational Chemistry Legacy Software

Like most research centers, NASA Ames has a number of very capable codes that do not fit the object model. However, it is often possible to create a c++ object that 'encapsulates' the legacy software. That is, the c++ object has methods that reformat their parameters, execute the legacy software, reformat the result and return it. When the legacy software does IO, the encapsulating object must intervene between the legacy software and the CORBA system. This technique allows existing codes to operate within an object oriented framework with minimal modification.

Agent Style Communication

In this context, agent software means software components that communicate by sending programs to each other. When each component is controlled by a command language, this is relatively easy to implement. Thus, a user interface component could control the tcl/c++ design software by writing a tcl command file and sending it to the design software for execution. This approach to software is powerful but not yet well understood.

Conclusions

The NanoDesign software is intended to design and test fullerene based hypothetical molecular machines and components. The system is in an early stage of development. Presently, tcl provides an interpreted interface, c++ objects represent design components, and a parallelized FORTRAN program simulates the machine. In the future, an architecture based on distributed objects is envisioned. A key requirement for this vision is a standard set of interfaces to various computational chemistry capabilities (e.g., force fields, integrators, etc.). A standard set of interfaces would allow vendors to supply small, high quality components to a distributed system. If you're interested in helping establish these standards, please contact the author at globus@nas.nasa.gov.

References

To companion papers.

NanoDesign: FORTRAN index files

The FORTRAN file tube.f from the example in the paper will contain:

```
INTEGER*4 tube(2)
tube(1) = 1
tube(2) = 616
```

This case is so simple that the FORTRAN indices are not very useful, but a more complex FORTRAN file for two gears with teeth might look like:

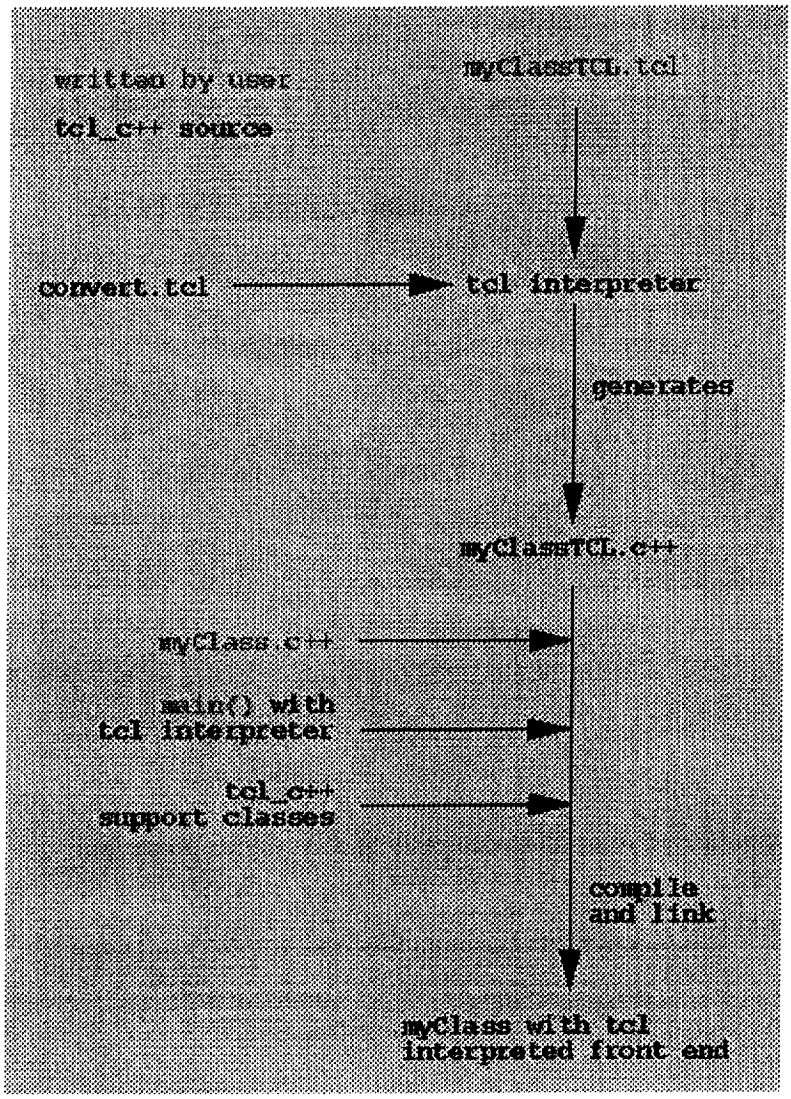
```
INTEGER*4 gear1(2)
INTEGER*4 tube1(2)
INTEGER*4 teeth1(2)
INTEGER*4 gear2(2)
INTEGER*4 tube2(2)
INTEGER*4 teeth2(2)
gear1(1) = 1
gear1(2) = 686
tube1(1) = 1
tube1(2) = 616
teeth1(1) = 617
teeth1(2) = 686
gear2(1) = 687
gear2(2) = 1372
tube2(1) = 687
tube2(2) = 1302
teeth2(1) = 1303
teeth2(2) = 1372
```

This information is necessary for the molecular dynamics to treat different parts of the system properly.

To the main body of the paper.

Web Work: Al Globus

tcl_c++: an Interface Between tcl and c++



c++_tcl architecture.

This software allows a c++ programmer to create a tcl interface to c++ objects with minimal coding overhead (basically one line per member function). The name of the class becomes a tcl procedure (with or without arguments depending on what constructors are defined) that calls the appropriate constructor and creates a tcl procedure to interface to the newly created object. This tcl procedure uses its first argument to decide which member function of the object to call. The value of variables and constants are also available to tcl programs. Multiple inheritance is supported.

To use tcl_c++ you must create a file called myClassTCL.tcl for each c++ class, for example:

```
startClass  myClass
constructor
constructor int double
memberFunction void member
```

```
memberFunction int integer
memberFunction int integerArgument int
memberFunction myClass* pointerArgumentReturn myClass*
memberFunction myClass& referenceArgumentReturn myClass&
memberFunction void twoArguments int char*
memberFunction int* handlePointers double*
memberConstant double  doubleConstant
memberVariable  int  integerVariable
destructorCommand delete
baseClass baseClass1
baseClass baseClass2
endClass
```

Note: only the parts of the class that should be available to tcl are put in the file.

This file is actually a tcl program. The tcl procedures startClass, constructor, memberFunction and so forth are defined in a file called convert.tcl. Now for the explanations:

- **startClass myClass** -- the class is named "myClass". This must always come first.
- **constructor** -- there is a constructor with no arguments. If no constructors are specified, this default constructor is created **unless** the tcl variable abstractClass is non-zero. In this case no constructors are allowed.
- **constructor int double** -- there is a constructor with two arguments, an int and a double. One might say: "set object [myClass 1 2.4]". Note that $object will be a number -- specifically the value of a pointer pointing to the object created. $object will also be a tcl procedure whose first argument will determine which member function, constant, or variable will be accessed.
- **memberFunction void member** -- this creates an interface to a member function named "member" with no arguments and no return value. Note that at present member function dispatch is on name and **number of arguments** only -- not type. This could be fixed by changing convert.tcl output to optionally look for lists containing the type followed by the value each time a member function is called.
- **memberFunction int integerFunction** -- this member function returns an integer.
- **memberFunction int integerArgument int** -- this one also takes an integer as an argument.
- **memberFunction myClass* pointerArgumentReturn myClass*** -- this function takes a pointer as an argument and returns a pointer.
- **memberFunction myClass& referenceArgumentReturn myClass&** -- this one handles reference arguments and return values (the return value is just a number that is the address of the returned object). This only works if myClass is derived from the tcl_c++ support class aTclCapableClass. aTclCapableClass doesn't do anything or allocate any memory, but it's necessary as a base class so the tcl_c++ generated code handles types properly.
- **memberFunction void twoArguments int char*** -- this member function takes two arguments. Note that char* has no spaces. This is necessary to have tcl consider it one argument. Alternatively, one could use "char *" (enclose it in double quotes).
- **memberFunction int* handlePointers double*** -- this function takes a pointer to a double as an argument (probably returned by a memberFunction) and returns a pointer to an integer. As far as tcl is concerned, the return value and argument are simply numbers. C++ knows that these numbers are pointer values.
- **memberConstant double doubleConstant** -- this creates an interface so tcl can get the value of a constant. The code might look like "set value [$object doubleConstant]". This works for static and non-static member constants.

- **memberVariable int integerVariable** -- same as memberConstant. It will only return the value. To set the value you must write a member function. This works for static and non-static member variables.
- **destructorCommand delete** -- "$object delete" will call the myClass destructor and tclDeleteCommand (see below). This is not well tested.
- **baseClass baseClass1** -- myClass has a base class accessible from tcl.
- **baseClass baseClass2** -- myClass has a second base class accessible from tcl. Member functions from baseClass1 will be used if there's a name conflict because baseClass1 is first. At present, the baseClass calls **must** come at the end.
- **endClass** -- signals the end of the class interfaces. This must always come last.

Note the following issues:

- tcl_c++ assumes that there is an include file called myClass.hh. If not, use "set includeFileName actual-include-file-name" before the startClass call. If includeFileName == "none", no class specific #include will be in the c++ file.
- To put custom code in the generated c++ file, use "puts" anywhere in your myClassTCL.tcl file.
- It's best if the classes you interface to are derived from aTclCapableClass. aTclCapableClass has no members so it should cause no problems as a base class. If you don't use aTclCapableClass as a base class, you won't be able to pass or return references to your class (pointers should work though). Note that classes without aTclCapableClass as a base class haven't been well tested.
- Pointers to most pre-defined types will have tcl names equal to their address. Char* is an exception, these are always handled as constant strings.
- Every class automatically has a tcl "member function" called tclDeleteCommand. When this is called, the tcl command created for the object will be replaced with an error message. This allows tcl to forget about objects.
- If a member function returns an object that does not have a tcl command, the tcl program won't be able to access this object's members. This can be fixed by using the automatically generated tcl procedure "cast2myClass" with one argument -- a pointer to the object (presumably returned by a member function). cast2myClass creates a tcl procedure to interface to the object, which is assumed to be an instance of myClass. Note that there is no type checking.

Known problems:

- Const has never been tested in any context.
- Unsigned anything and long double arguments and return values aren't supported.
- Operators and non-member functions, constants and variables aren't supported.
- tcl_c++ doesn't check to see if input numbers are really numbers.
- tcl_c++ isn't thread compatible.
- The build generates a number of unimportant c++ error messages.
- tcl_c++ puts a few name constraints on your c++ software. Namely, there can be no class members named aTclCommand_argument followed by a number. Also, there may be no functions called _myClass__handleMemberFunctionsTclCommand. These restrictions shouldn't be too severe.
- There is one constraint on the names of tcl procedures you can use, cast2myClass is automatically created for each class.
- Memory management issues haven't been carefully address or tested. Only the c++ delete operator is directly supported, not reference counting schemes. One can call a member function and then

use tclDeleteCommand though.
- Tcl_c++ has only been tested on IRIX 5.3.

I had hoped to distribute the code on the WWW, but it's not quite ready for wide distribution -- especially the regression test. If you are interested in using tcl_c++, send email to globus@nas.nasa.gov.

To the NanoDesign paper.

Author: Al Globus

References

[Allen 87] M. P. Allen, D. J. Tildesley, *Computer Simulation of Liquids*, Oxford Science Publications (1987).

[Bauschlicher 96] Charles W. Bauschlicher, Jr., Ralph Merkle, "The Chemical Storage of Data," submitted to *Nanotechnology* (1996).

[Brenner 90] Don W. Brenner, *Phys. Rew. B* 42, 9458 (1990).

[Cluzel 96] Philippe Cluzel, Anne Lebrun, Christoph Heller, Richard Lavery, Jean-Louis Viovy, Didier Chatenay, Francois Caron, "DNA: An Extensible Molecule," *Science* , volume 271, 9 February 1996, pp. 792-794.

[Diederich 96] F. Diederich and C. Thilgen, "Covalent Fullerene Chemistry," *Science*, volume 271, 19 January 1996, pp. 317-323.

[Dresselhaus 96] M. S. Dresselhaus, G. Dresselhaus, P. C. Eklund, *Science of Fullerenes and Carbon Nanotubes*, Academic Press (1996).

[Drexler 92a] K. Eric Drexler, *Nanosystems: Molecular Machinery, Manufacturing,and Computation*, John Wiley & Sons, Inc. (1992).

[Drexler 1992b] K. Eric Drexler, *Journal of the British Interplanetary Society*, volume 45, number 10, pp. 401-405 (1992).

[Ebbesen 96] Thomas W. Ebbesen, "Carbon Nanotubes," *Phys. Today* June 27 (1996).

[Freemantle 96] Michael Freemantle, "Filled Carbon Nanotubes Could Lead to Improved Catalysts and Biosensors," *C&EN* July 15 (1996).

[Gao 96] Guanghua Gao, Tahir Cagin, William A. Goddard, III, Al Globus, Ralph Merkle, K. Eric Drexler, "Molecular Dynamics Simulations of Molecular Planetary Gears," *First Electronic Molecular Modelling & Graphics Society Conference* (1996).

[Genesereth 94] Michael R. Genesereth, Steven P. Ketchpel, "Software Agents," CACM 37 (7): 48-53, 147 (1994).

[Girifalco 92] L. A. Girifalco, "Molecular Properties of C60 in the Gas and Solid Phases," *J. Phys. Chem.* 96, 858 (1992)

[Globus 96a] Al Globus, Richard Jaffe, "NanoDesign: Concepts and Software for a Nanotechnology Based on Functionalized Fullerenes," *First Electronic Molecular Modelling & Graphics Society Conference* (1996).

[Globus 96b] Al Globus, David Bailey, Steve Langhoff, Andrew Pohorille, and Creon Levit, "Computational Nanotechnology at NASA Ames Research Center, 1996," *First Electronic Molecular*

Modelling & Graphics Society Conference (1996).

[Han 96] Jie Han; Al Globus, Richard Jaffe, Glenn Deardorff, "Molecular Dynamics Simulation of Carbon Nanotube Based Gears," *First Electronic Molecular Modelling & Graphics Society Conference* (1996).

[Hoke 92] Steven H. Hoke, Jay Molstad, Dominique Dilettato, Mary Jennifer Jay, Dean Carlson, Bart Kahr and R. Graham Cooks, "Reaction of Fullerenes and Benzyne," *Journal of Organic Chemistry*, 11 September 1992, V57 N19:5069-5071.

[Jaffe 96a] Richard Jaffe, Jie Han, and Al Globus, "Formation of Carbon Nantoube Based Gears: Quantum Chemistry and Molecular Dynamics Simulations of the Electrophilic Addition of o-Benzyne to Fullerenes, Graphene, and Nanotubes," *First Electronic Molecular Modelling & Graphics Society Conference* (1996).

[Jaffe 96b] Richard L. Jaffe, Grant D. Smith, "A Quantum Chemistry Study of Benzene Dimer," *J. Chem. Phys,* 105, 2780 (1996)

[Jung 96] T. A. Jung, R. R. Schlittler, J. K. Gimzewski, H. Tang, C. Joachim, "Controlled Room-Temperature Positioning of Individual Molecules: Molecular Flexure and Motion," *Science* 271, 181 (1996).

[McKendree 95] Tom McKendree, "Implications of Molecular Nanotechnology: Technical Performance Parameters on Previously Defined Space System Architectures," The Fourth Foresight Conference on Molecular Nanotechnology, Palo Alto, CA. (November 1995).

[Ousterhout 94] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, publisher (1994).

[Pearson 75] Jerome Pearson, *Acta Astronautica* **2** pp. 785-799 (1995).

[Rinzler 95] A. G. Rinzler, J. H. Hafner, P. Nikolaev, L, Lou, S. G. Kim, D. Tománek, P. Nordlander, D. T. Colbert, R. E. Smalley, "Unraveling Nanotubes: Field Emission From an Atomic Wires," *Science* 269, 1550 (1995).

[Robertson 92] D. H. Robertson, D. W. Brenner, J. W. Mintmire, "Energetics of Nanoscale Graphite Tubules," *Phys. Rev.* **B**45, 12592 (1992).

[Saini 96] Subhash Saini, "Petaflop Computing and Computational Nanotechnology", Nanotechnology, 1996 (in press).

[Schroeder 95] Will Schroeder, Ken Martin, Bill Lorensen, *The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics*, ISBN 0-13-199837-4, Prentice Hall, (February 1996).

[Service 96] Robert F. Service, "Mixing Nanotube Structures to Make a Tiny Switch," *Science* 271, 1232 (1996).

[Siegel 96] Jon Siegel, Dan Frantz, Hal Mirsky, Raghu Hudli, Peter de Jong, Alan Klein, Brent Wilins, Alex Thomas, Wilf Coles, Sean Baker, Maurice Balick, *CORBA Fundamentals and Programming*,

Object Management Group, Wiley Computer Publishing group, John Wiley & Sons, Inc. (1996).

[Smith 96a] Steven B. Smith, Yujia Cui, Carlos Bustamante, "Overstretching B-DNA: The Elastic Response of Individual Double-Stranded and Single-Stranded DNA Molecules," *Science*, volume 271, 9 February 1996, pp. 795-799.

[Smith 96b] Grant D. Smith, Richard Jaffe, "Comparative Study of Force Fields for Benzene," *J. Phy. Chem.* 100, 9624 (1996)

[Stroustrup 91] B. Stroustrup, *The C++ Programming Language*, second edition, Addison-Wesley (1991).

[Thess 96] Andreas Thess, Roland Lee, Pavel Nikolaev, Hongjie Dai, Pierre Petit, Jerome Robert, Chunhui Xu, Young Hee Lee, Seong Gon Kim, Andrew G. Rinzler, Daniel T. Colbert, Gustavo E. Scuseria, David Tomanek, John E. Fischer, Richard E. Smalley, "Crystalline Ropes of Metallic Carbon Nanotubes," *Science*, volume 273, 26 July 1996, pp. 483-487.

[Treacy 96] M. M. J. Treacy, T. W. Ebbesen, J. M. Gibson, "Exceptionally High Young's Modulus Observed for Individual Carbon Nanotubes," *Nature* 381, 678 (1996).

[Tuzun 95] Robert E. Tuzun, Donald W. Noid, and Bobby G. Sumpter, "Dynamics of a Laser Driven Molecular Motor," *Nanotechnology* 6, pp. 52-63 (1995).

[Yazdani 96] Ali Yazdani, D. M. Eigler, N. D. Lang, "Off-Resonance Conduction Through Atomic Wires," *Science*, volume 272, 28 June 1996, pp. 1921-1923.

To papers.

Web Work: Al Globus