Navigation Ancillary Information Facility

The goal of the Navigation and Ancillary Information Facility is to provide the planetary science community with datasets and transportable software tools, appropriate for computing, archiving, accessing and distributing the ancillary viewing geometry needed to interpret observations of solar system bodies.

Amphion: Automatic Programming for the NAIF Toolkit

Michael Lowry, Andrew Philpot, Thomas Pressburger, Ian Underwood, Ames Research Center, Richard Waldinger, Mark Stickel, SRI International

Suppose you are a planetary scientist working on observations for Galileo's tour. You want to compute the predicted solar incidence angle at the sub-spacecraft point of Galileo on Jupiter. You have some knowledge of the SPICE system and its NAIF Toolkit, but you haven't used it much.

If you describe the problem like this:

Let x be the angle between two rays, u and v.

Let p be the point on the "surface" of Jupiter closest to Galileo at time t.

Let u be the ray from a point p to the Sun.

Let v be the "surface" normal at p.

(In SPICE Jupiter is normally modeled as a triaxial ellipsoid with a "surface" at the 1 bar pressure level.)

Let the representation of angle x be in radians.

Let the representation of time t for Galileo be a string.

then you might construct the program named SOLAR shown in Figure 1. This program makes the appropriate calls to SPICELIB, the subroutine library of the NAIF Toolkit.

Alternatively, you could request Amphion, an automatic programming system developed by the Artificial Intelligence Branch at Ames Research Center, to write the program for you. In this SPICE application Amphion takes specifications for geometric problems and returns FORTRAN code that calls NAIF Toolkit subroutines and functions. In fact, Amphion wrote the SOLAR program of Figure 1.

Amphion consists of a program synthesis component and a graphical user interface front end. The program synthesis component takes as input a specification written in a textual form conceptually similar to the one above. The output is a FORTRAN program, such as the SOLAR program. However, instead of writing text, you enter specifications graphically through a menu-guided interface. This interface translates completed specification diagrams such as the SOLAR specification in Figure 2 to the textual form used by the program synthesis component.

Amphion's specification language is at the level of abstract geometry—note that the specification above is purely geometric: there is no mention of coordinate frames, units, and so on, except in defining conventions for inputs and outputs. The vocabulary is basic Euclidean geometry (e.g. points, rays, ellipsoids, and intersections) augmented with astronomical terms (e.g. photons and ephemeris objects).

How would you learn the details of the vocabulary and syntax of Amphion's specification language? The graphical user interface (GUI) makes this unnecessary. The GUI allows you to enter specifications as graphs of geometric objects and constraints, so you don't need to know the syntax of the specification language. It

```
      SUBROUTINE SOLAR ( GALILE, ANGLEI)

C     Input Parameters
      CHARACTER*(*) GALILE

C     Output Parameters
      DOUBLE PRECISION ANGLEI

C     Function Declarations
      DOUBLE PRECISION VSEP

C     Parameter Declarations
      INTEGER JUPITE
      PARAMETER (JUPITE = 599)
      INTEGER GALILE1
      PARAMETER (GALILE1 = -77)
      INTEGER SUN
      PARAMETER (SUN = 10)

C     Variable Declarations
      DOUBLE PRECISION RADJUP ( 3 )
      DOUBLE PRECISION E
      DOUBLE PRECISION PVGALI ( 6 )
      DOUBLE PRECISION LTJUGA
      DOUBLE PRECISION V1 ( 3 )
      DOUBLE PRECISION X
      DOUBLE PRECISION PVJUPI ( 6 )
      DOUBLE PRECISION LTSUJU
      DOUBLE PRECISION MJUPIT ( 3, 3 )
      DOUBLE PRECISION V2 ( 3 )
      DOUBLE PRECISION X1
      DOUBLE PRECISION DV2V1 ( 3 )
      DOUBLE PRECISION PVSUN ( 6 )
      DOUBLE PRECISION XDV2V1 ( 3 )
      DOUBLE PRECISION V ( 3 )
      DOUBLE PRECISION N ( 3 )
      DOUBLE PRECISION PN ( 3 )
      DOUBLE PRECISION DV2N ( 3 )
      DOUBLE PRECISION XDV2N ( 3 )
      DOUBLE PRECISION DXDV2V ( 3 )
      DOUBLE PRECISION XDXDV2 ( 3 )

C     Dummy Variable Declarations
      INTEGER DMY10
      DOUBLE PRECISION DMY20 ( 6 )
      DOUBLE PRECISION DYM60 ( 6 )
      DOUBLE PRECISION DMY130

      CALL BODVAR ( JUPITE, 'RADII',
      DMY10, RADJUP )
      CALL SCS2E ( GALILE1, GALILE, E )
      CALL SPKSSB ( GALILE1, E, 'J2000',
      PVGALI )
      CALL SPKEZ ( JUPITE, E, 'J2000',
      'NONE', GALILE1, DMY20, LTJUGA )
      CALL VEQU ( PVGALI ( 1 ), V1 )
      X = E - LTJUGA
      CALL SPKSSB ( JUPITE, X, 'J2000',
      PVJUPI )
      CALL SPKEZ ( SUN, X, 'J2000',
      'NONE', JUPITE, DMY60, LTSUJU )
      CALL BODMAT ( JUPITE, X, MJUPIT )
      CALL VEQU ( PVJUPI ( 1 ), V2 )
      X1 = X - LTSUJU
      CALL VSUB ( V1, V2, DV2V1 )
      CALL SPKSSB ( SUN, X1, 'J2000',
      PVSUN )
      CALL MXV ( MJUPIT, DV2V1, XDV2V1 )
      CALL VEQU ( PVSUN ( 1 ), V )
      CALL NEARPT ( XDV2V1, RADJUP ( 1 ),
      RADJUP ( 2 ), RADJUP ( 3 ),.N,
      DMY130 )
      CALL SURFNM ( RADJUP ( 1 ), RADJUP
      ( 2 ), RADJUP ( 3 ), N, PN )
      CALL VSUB ( N, V2, DV2N )
      CALL MTXV ( MJUPIT, DV2N, XDV2N )
      CALL VSUB ( V, XDV2N, DXDV2V )
      CALL MXV ( MJUPIT, DXDV2V, XDXDV2 )
      ANGLEI = VSEP ( XDXDV2, PN )

      RETURN
      END
```

Figure 1. SOLAR program corresponding to the diagram in Figure 2. All the subroutines and functions in the SOLAR program are from the standard NAIF Toolkit.

also guides you in the process: pop-up menus display what is possible to do next, given what you have already done.

Amphion employs an object-oriented paradigm for interactively developing problem specifications. Conceptually, you develop a problem specification by first defining a configuration, and then declaring a subset of the objects in a configuration to be inputs or outputs of the desired FORTRAN program. A configuration is a set of objects and their relationships.

A configuration is generated through the actions of adding objects, deleting objects, moving the edges between objects that define their interrelationships, and by merging objects together. Adding and deleting objects are done through pop-up menus; moving edges and merging objects are done by directly manipulating the diagram. Declaring an object to be an input or output brings up a menu of the possible data conventions: coordinate systems for locations, time systems for time, and units of measurement. After a specification is developed, Amphion performs checks for consistency and completeness. If a specification is faulty, Amphion gives you appropriate feedback; for example, by telling you that a particular object in the diagram is under constrained with respect to the given inputs. If a specification is good, Amphion generates the text of a FORTRAN program for you within a few minutes.

Programming at the specification level

The objective of Amphion is to enable you to program at the level of abstract domain-oriented problem specifications, rather than at the detailed level of subroutine calls. The idea is that you should be able to use the library without having to absorb all the documentation. You should also be able to reuse previous specifications that are similar to yours without having to read that specification's code. This is where Amphion really comes into its own:

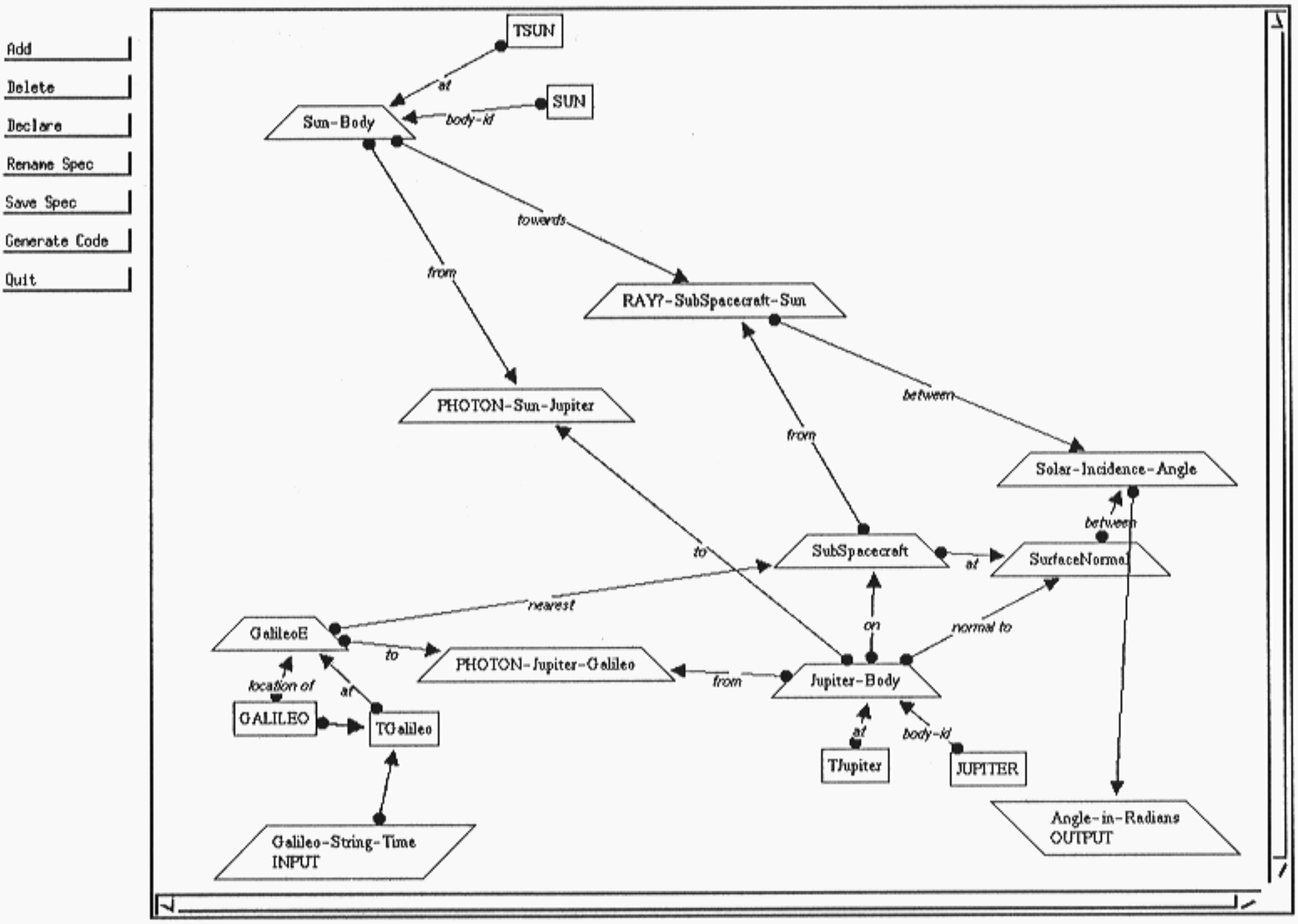

Figure 2. The graphical specification for solar incidence angle developed interactively with Amphion's front end.

- Previous specifications can be used as a starting point, instead of starting from scratch.
- The abstract graphical notation makes it much easier to identify the required modifications than tracing through dependencies in someone's code. Amphion's editing operations facilitate making the changes.
- There is no possibility of introducing bugs in the code, since Amphion synthesizes the code from scratch for the modified specification.

For example, if you wanted to modify the specification above to be the solar incidence angle at the point on the surface at the center of an instrument boresight, you would just do the following steps:

1. Add the instrument and the ray along the boresight.
2. Add the intersection point of this ray with the surface.
3. Replace the sub-spacecraft point with this intersection point.

With Amphion it is easy to make a wide variety of modifications to this specification, such as:

- Make the names of the spacecraft and the planet variables that are input to the program.
- Add more outputs, such as the location of the sub-spacecraft point in Jupiter's planetocentric coordinate system.
- Change the input and output representations (e.g. different time systems and coordinate systems).
- Use the sub-diagram for the sub-spacecraft point as a component in another diagram.
- Change the sub-spacecraft point and the planet into input variables, and then use the appropriate sub-diagram as a general specification for a solar incidence angle program.

The last modification is especially interesting, because it describes one quantity computed by a new Toolkit subroutine named ILLUM_M. The other quantities computed by ILLUM_M are emission and phase angle, which can be added to the diagram to make a full specification of the ILLUM_M subroutine. Note that the same interface for specifying problems could also be used to define subroutines to add to Amphion's knowledge base. This will be important in the future as the NAIF Toolkit is expanded, enabling the members of the Jet Propulstion Laboratory's NAIF group to maintain Amphion themselves. (ILLUM_M is not yet part of the standard NAIF Toolkit but is available on request from NAIF.)

Current status

Amphion has been installed at the Jet Propulsion Laboratory and is being tested by members of the Navigation and Ancillary Information Facility (NAIF) team. In preliminary testing with a prototype in the fall of 1993, scientists were able to develop their own specifications after a one-hour tutorial. Both the GUI and the program synthesis components are driven by the same declarative domain theory, ensuring compatibility between these two components. Amphion's programs are guaranteed to be correct implementations of specifications (with respect to the domain theory), because they are generated as a side effect of a proof process that establishes their correctness. The proofs are generated by the SNARK automatic theorem prover from SRI, developed by Mark Stickel and modified by Richard Waldinger for the purpose of program synthesis. The current domain theory is incomplete, but covers most of the core data access and geometry routines in the NAIF Toolkit.

Some aspects of the domain theory development and extension were relatively straightforward. Extensions that only define new subroutines are particularly easy and after further development will likely be made directly by the NAIF group using the Amphion interface, as described in the ILLUM_M example above. Even extensions that require defining new kinds of abstract objects are usually easy to make. For example, during a visit in September 1993, NAIF Task Manager Chuck Acton wanted to specify a problem involving a new kind of object that was not yet part of the domain theory: the pole of a planet. In 15 minutes we were able to add into the domain theory the declarations and axioms that were needed. Then Amphion was able to generate a FORTRAN program for a specification involving the pole of a planet.

However, some aspects of domain theory development can be difficult. For example, adding in the facility for declaring different coordinate systems (e.g., spherical, rectangular, planetocentric) to the previously existing facility for declaring different coordinate frames for input and output required changes in many existing axioms.

Future Plans

Our plans for this coming year are to develop Amphion into a robust tool that can be distributed to user sites with the NAIF Toolkit for alpha testing. We will extend the domain theory to provide wider coverage of the NAIF Toolkit. We will also enable Amphion to generate sophisticated programs, such as iterative programs that search for occurrences of a geometric configuration (e.g. an occultation) within a user-specified time window.

A major area of further development will be the front-end interface for specification development. To make it easier to reuse specifications, they will be indexed so that relevant predecessors can be retrieved through a dialog about a user's requirements. Features will be added to make it easier to understand specifications created by other users. Users will also be able to encapsulate small, parameterized specifications as definitions that will be added to the domain theory.

Why the name Amphion? Amphion was the son of Zeus who used his magic lyre to charm the stones lying around Thebes into position to form the city's walls. The Amphion system's expertise lies in charming subroutines into useful programs through an advanced automatic theorem prover. Because it uses a generic architecture Amphion could be applied to other libraries by developing the appropriate domain theory. Our long-range goal is to expand Amphion into a full generic automatic programming system that empowers domain experts to develop and maintain domain theories with only occasional consultation from experts in automatic programming.

For further information on Amphion, including discussion of possibilities for new applications, contact Michael Lowry at:
(408) 604-3369; lowry@pluto.arc.nasa.gov