AMPHION: Specification-Based Programming for Scientific Subroutine Libraries

N95-23680

Michael Lowry, Andrew Philpot, Thomas Pressburger, and Ian Underwood
Recom Technologies; AI Research Branch, M.S. 269-2
NASA Ames Research Center
Moffett Field, CA 94305, USA
Tel: (415) 604-3369 Fax: (415) 604-3594
lowry@ptolemy.arc.nasa.gov

Richard Waldinger and Mark Stickel
AI Center; SRI International

KEY WORDS AND PHRASES

Artificial Intelligence, knowledge-based software engineering, NAIF, software engineering, software reuse.

OVERVIEW

AMPHION is a knowledge-based software engineering (KBSE) system that guides a user in developing a diagram representing a formal problem specification. It then automatically implements a solution to this specification as a program consisting of calls to subroutines from a library. The diagram provides an intuitive domain-oriented notation for creating a specification that also facilitates reuse and modification.

AMPHION'S architecture is domain independent. AMPHION is specialized to an application domain by developing a declarative domain theory. Creating a domain theory is an iterative process that currently requires the joint expertise of domain experts and experts in automated formal methods for software development.

AMPHION has been applied to JPL's NAIF domain through a declarative domain theory that includes an axiomatization of JPL's SPICELIB subroutine library. Testing with planetary scientists demonstrates that AMPHION's interactive specification acquisition paradigm enables users to easily develop, modify, and reuse specifications after only a short tutorial. AMPHION routinely synthesizes programs consisting of dozens of SPICELIB subroutine calls from these specifications in just a few minutes.

Qualitative assessments indicate an order of magnitude productivity increase using AMPHION over manual program development. AMPHION is currently undergoing alpha testing in preparation for distribution to the NAIF community. Other NASA domains are under consideration. Future research will address the technology needed for domain experts to develop their own AMPHION domain theories with only minimal consultation from experts in formal methods.

MOTIVATION

Within the space science community, subroutine libraries are a ubiquitous form of software reuse. However, space scientists often do not make effective use of libraries. Sometimes this happens because a subroutine library is developed without following good conventional software engineering practices, resulting in inadequate documentation, untrustworthy code, and a lack of coherence in the different functions performed by the individual routines. However, even when a subroutine library is developed following the best conventional software engineering practices, users often have neither the time nor the inclination to fully familiarize themselves with it. The result is that most users lack the expertise to properly identify and assemble the routines appropriate to their problems. This represents an inherent knowledge barrier that lowers the utility of even the best-engineered software libraries: the effort to acquire the knowledge to effectively use a subroutine library is often perceived as being more than the effort to develop the code from scratch. AMPHION is an effective solution to this knowledge barrier.

The objective of AMPHION is to enable users who are familiar with the basic concepts of an application domain to program at the level of ab-

stract domain-oriented problem specifications, rather than at the detailed level of subroutine calls. AMPHION breaks through the knowledge barrier by enabling use of a subroutine library without having to absorb all the documentation about a library, especially the plethora of implementation details such as the representation conventions for subroutine parameters.

NAIF APPLICATION

The first application domain for AMPHION is solar-system kinematics, as implemented in the SPICELIB subroutine library developed by the Navigation Ancillary Information Facility (NAIF) at JPL. SPICELIB is an extremely well-engineered library used by planetary scientists to plan and analyze the observing geometry for data collected during interplanetary missions or by space-based telescopes. A domain theory was developed that includes an abstract formalization of solar-system kinematics suitable for specifying problems, and the knowledge needed to implement solutions using SPICELIB. To date, Amphion has demonstrated the following essential capabilities for real-world KBSE:

1. Users without training in formal methods readily develop domain-oriented diagrams corresponding to formal problem specifications using Amphion's specification-acquisition tools.
2. Users can reuse, modify, and maintain previously developed specifications, thereby elevating the software life cycle from the code level to the specification level.
3. Automatic deductive program synthesis achieves acceptable performance, given an appropriately structured domain theory and moderate use of theorem-proving tactics.

Programming at the Specification Level

To enable users to program at the specification level, AMPHION consists of a specification-acquisition component to guide users in developing a formal specification, and a program synthesis component that automatically generates a program implementing a solution to the specification. Users enter specifications graphically through a menu-guided graphical user interface (GUI). Figure 1 is an example of a completed specification: it denotes the problem of predicting the solar incidence angle at the point on Jupiter closest to Galileo at a particular time. (This is the sub-spacecraft point). The specification acquisition component performs semantic checks on completed specification diagrams, and then automatically translates them to a logical form used by the program synthesis component.

The output of program synthesis for the NAIF application is a FORTRAN–77 program consisting of calls to the SPICELIB subroutine library. AMPHION generated the SOLAR program in Figure 2 from the specification in Figure 1 in 52 seconds of CPU time on a Sparc 2. In over a hundred programs generated by AMPHION for the NAIF domain to date, the CPU time has exceeded three minutes in only four cases. This is an unprecedented level of performance for the deductive synthesis approach, developed over 25 years ago [1,2]. Most of the program synthesis component is independent of the target output language. It would only take two weeks of work to adapt AMPHION for a different output language such as C or UNIX shell files.

AMPHION's specification language for the NAIF domain is at the level of abstract geometry. This specification language is part of the declarative domain theory. The vocabulary is basic Euclidean geometry (e.g., points, rays, ellipsoids, and intersections) augmented with astronomical terms (e.g., planets, spacecraft, and photons; the latter for specifying constraints used in calculating light-time correction). The specification language does not include the myriad implementation details required for correctly calling SPICELIB subroutines, such as coordinate frames, units, time systems, etc; these details are automatically deduced during program synthesis. The user only needs to define the abstract problem and the desired representation conventions for the program inputs and outputs.

AMPHION's GUI bears a superficial resemblance to data-flow oriented graphical programming environments. For example, Apple's HOOKUP application enables users to select icons from a palette that represent individual subroutines, and then connect input and output ports. However, these environments only provide an alternate notation to conventional programming languages. In contrast, AMPHION enables a radical separation between the level at which users

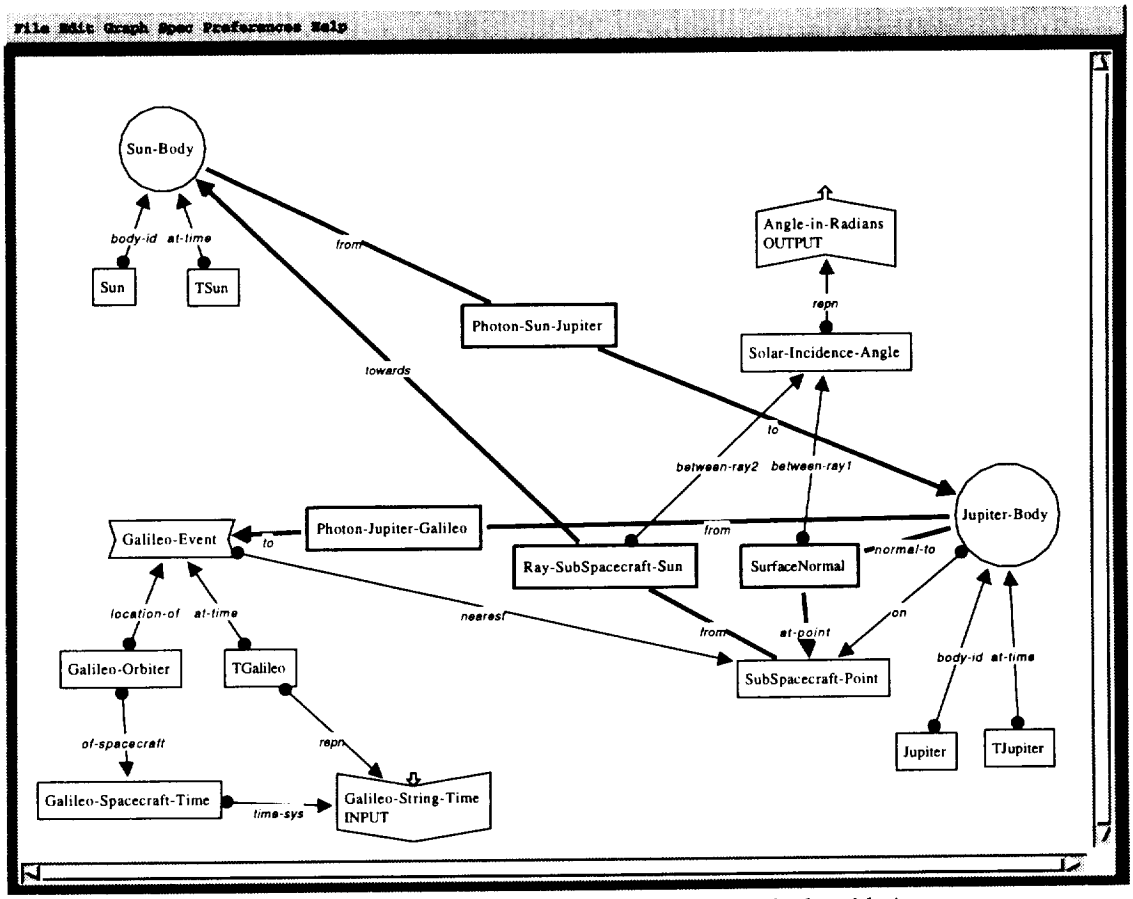

Figure 1: Diagram for solar incidence angle developed interactively with AMPHION.

```
      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 GALIL1
      PARAMETER (GALIL1 = -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 DMY60 ( 6 )
      DOUBLE PRECISION DMY130
      CALL BODVAR ( JUPITE, 'RADII', DMY10, RADJUP )
      CALL SCS2E ( GALIL1, GALILE, E )
      CALL SPKSSB ( GALIL1, E, 'J2000', PVGALI )
      CALL SPKEZ ( JUPITE, E, 'J2000', 'NONE', GALIL1,
     .             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 2: SOLAR program generated by AMPHION from Figure 2.

specify problems and the level at which solutions are implemented by the program synthesis component. AMPHION's GUI provides an alternate notation to formal specifications written in mathematical logic. The notation of mathematical logic can be formidable; that is one reason that specification-based software engineering life cycles have not previously been adopted in practice.

AMPHION's GUI employs an object-oriented paradigm for interactively developing problem specifications. Conceptually, a user develops 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 program. A configuration is a set of abstract objects and their relationships.

A user generates a configuration 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 menus; moving edges and merging objects are done by directly manipulating the diagram. Declaring an object to be an input or output of the desired program brings up a menu of the possible data-representation conventions: coordinate systems for locations, time systems for time, and units of measurement. These alternative representation conventions are also part of the declarative domain theory.

AMPHION's specification-acquisition component not only enables specifications to be developed from scratch, but it is also especially well suited for specification reuse and modification. The abstract graphical notation makes it much easier to identify the required modifications than it is to trace through dependencies in code. AMPHION's editing operations facilitate making the changes. Furthermore, there is no possibility of introducing bugs in the code, since AMPHION synthesizes the code from scratch for the modified specification.

FUTURE DIRECTIONS

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 SNARK, an advanced automatic theorem prover developed at SRI International. A tutorial introduction for this deductive approach to program synthesis can be found in [3], while more details on the use of SNARK for synthesizing programs in the NAIF domain can be found in [4]. One advantage of the deductive approach is that a synthesized program is guaranteed to be a correct implementation of a user's specification, with respect to the domain theory. This reduces the software verification problem to a one-time verification of the domain theory. The declarative nature of the domain theory simplifies verification.

Because it uses a generic architecture, described in [5], AMPHION can be applied to other domains and subroutine libraries by developing the appropriate domain theories. The methodology for developing suitable AMPHION domain theories is described in [6]. Developing the initial NAIF domain theory took three months of collaboration between a NAIF expert and experts in automated formal approaches to program synthesis. Much of the subsequent refinements to the domain theory were straightforward and could likely be done by domain experts with the appropriate tools. Future research will include developing such tools.

REFERENCES

[1] Green, C.; 1969. "Application of Theorem Proving to Problem Solving", in IJCAI-69.

[2] Waldinger, R.; and Lee, R; 1969. "PROW: A Step Toward Automatic Program Writing", in IJCAI-69.

[3] Manna, Z.; and Waldinger, R.; 1992. "Fundamentals of Deductive Program Synthesis", in IEEE Transactions on Software Engineering (18) 8.

[4] Stickel, M.; Waldinger, R.; Lowry, M.; Pressburger, T.; and Underwood, I.; 1994. "Deductive Composition of Astronomical Software from Subroutine Libraries", in CADE-12.

[5] Lowry, M.; Pressburger, T.; Philpot, A.; and Underwood, I.; 1994. "A Formal Approach to Domain-Oriented Software Design Environments", in KBSE-94.

[6] Lowry, M.; Pressburger, T.; Philpot, A.; and Underwood, I.; 1994. "AMPHION: Automatic Programming for Scientific Subroutine Libraries", in ISMIS-94.

SESSION MD

Monitoring and Diagnostics