Today: Amphion System

- Computer Assisted SE system
- for the composition software library items

- Amphion system:
  developed by NASA by Automated Software Engineering group;
  work at NASA on program synthesis is described at

  `ti.arc.nasa.gov/ase/docs/progsyn.html`

- Putting together programs for domain specific tasks;
- Helps with the *program reuse* problem;
- Even well documented software libraries are under-used – how can the user
  – find out what is there?
  – put the routines together?

Methodology

- The user is guided in putting together a formal specification;
- A program meeting the specification is assembled automatically from a library of subroutines.

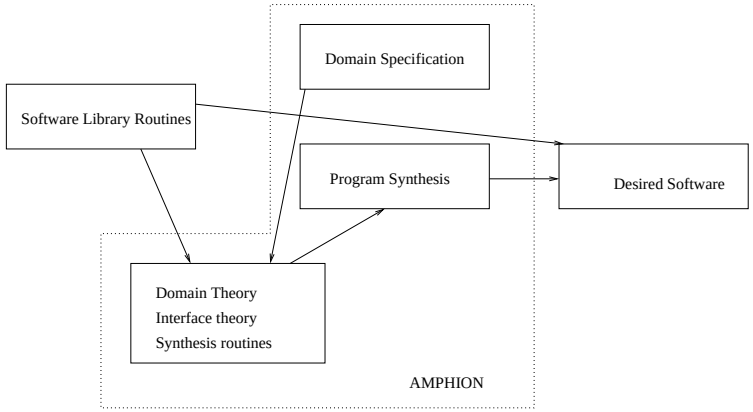
So users need to know about the basic concepts of the application domain; but *not* about the programming constructs.

Aimed at experts in the domain in question, without assuming programming expertise.

Components

- *Specification acquisition subsystem* (generic) –
  To build up domain characterisation.
- *Program synthesis subsystem* (generic) –
  To assemble programs from routines and overall specifications
- *Domain specific subsystem* –
  Theory of the domain, interface routines and automation routines for the synthesis system.

System Architecture



In use

- Used to develop routines to control astronomical observations;
- Builds on existing library of astronomical routines;
- Programs can be assembled much faster;
- Users can use the system after a short tutorial;
- Uses *deductive synthesis* to put together programs *automatically*.

Specifications

These are written in first order logic, augmented with lambda-calculus (like ML `fn x=> ...` syntax).

The shape of a specification is:

$\texttt{lambda(inputs)find(outputs)exists(intermed)}$
$\quad \texttt{conj}_1 \wedge \texttt{conj}_2 \cdots \wedge \texttt{conj}_n$

where the conjuncts are either an equation defining a function, or a constraint ($\texttt{P}(\texttt{v}_1, \ldots, \texttt{v}_m)$).

Specifications are checked to see that they can be solved abstractly. If they cannot be solved, error information is returned to the user.

Deriving the program

Given a satisfiable specification, look to find a proof of a statement:

$$\forall \texttt{inputs}\ \exists \texttt{outputs}\ \texttt{spec}(\texttt{inputs}, \texttt{outputs})$$

A derivation of a statement of this form gives directly a functional program that fits the specification, ie a program F such that

$$\forall \texttt{inputs}\ \texttt{spec}(\texttt{inputs}, \texttt{F}(\texttt{inputs}))$$

Use a functional language where the terms correspond to routines in the target (imperative) programming language; the functional program can get translated.

Language specific routines

Only at this point is something related to the programming language used; generate variable declarations and sequence of subroutine calls.

So, the system could be easily adapted to different programming languages.

Example problem

Suppose want to compute angle of sun light at a point on a planet's surface visible from satellite boresight. Specify this by a series of statements:

> Let *Solar-Incidence-Angle* be angle between rays *Surface-Normal* and *Ray-Intersection*.
>
> Let *Surface-Normal* be ray normal to *Jupiter-Body* at the point *Boresight-Intersection*.
>
> . . .

These are the conjuncts of the specification, in the language of Euclidean geometry. A diagrammatic representation is given to the user, and the program synthesised.

Domain Engineering

Amphion needs to know how to use the domain information (eg geometry).

The subroutines are assumed to be given, and correct w.r.t their own specification. So Amphion has

- abstract theory of domain (eg geometry);
- concrete theory of what the subroutines do;
- implementation relation between abstract and concrete theories.

Abstract Theory

This is an iterative process, involving the domain expert, *and* KBSE expert. For the astronomical domain, it involved collaboration over some time. For example, need here to choose time system, and notion of coordinate frame.

The abstract theory here has:

- types for objects, e.g. points, lines, ellipsoids, . . .
- constructors (eg ray from point and direction);
- geometric operations (eg intersection).

Abstract theory ctd

Also need some other relations between the concepts, eg

- the relation between times and places corresponding to light travelling from one to the other.

So, this is a fairly complex theory.

The existing routines have to be characterised also, in terms of programming language datatypes (vectors of reals, eg). Conversion functions are needed too.

Specification Acquisition

This builds on the domain theory syntax, via GUI, which also ensures the consistence of the input format.

Specifications may be put together in a top-down, or bottom-up style, according to taste.

The specification will be checked abstractly, to weed out some obvious problems; warnings are given of over-constrained and under-constrained variables. Some overloading of functions and relations is allowed.

Domain Theories and Specification

- Alongside descriptions of code capabilities, we want descriptions of the domain that the final system is intended for (astronomical observations, computer vision systems, telephone exchange software, etc.).

- Having built such a KB, we want also to support the construction of a specification of a particular problem specification in the domain.

We looks at ways in which these two tasks can be aided, with reference to the Amphion system and others.

Domain theories ctd

In each case, we assume there is an underlying logical formulation of the domain theory and of the problem specification. There are several ways we can be helped here:

- Provision of graphical representation;

- Consistency checking: is the domain theory consistent on its own?

- Well-formed input: checking that the specification makes logical sense (e.g. only uses syntax present in the domain theory).

This assumes that the task is specified in terms of the domain theory, and not in terms of computation — this is more acceptable to end users.

Graphical/Logical Translations

There are other places where there is such a close link between graphical description and logical that we can translate from one to the other.

For example, consider describing an *electrical circuit*; it is possible to build up circuit descriptions graphically, and get a corresponding logical description automatically, in terms of a fixed syntax such as one we saw (`andg`, `org`, . . . , `output`, `input`, `connected`).

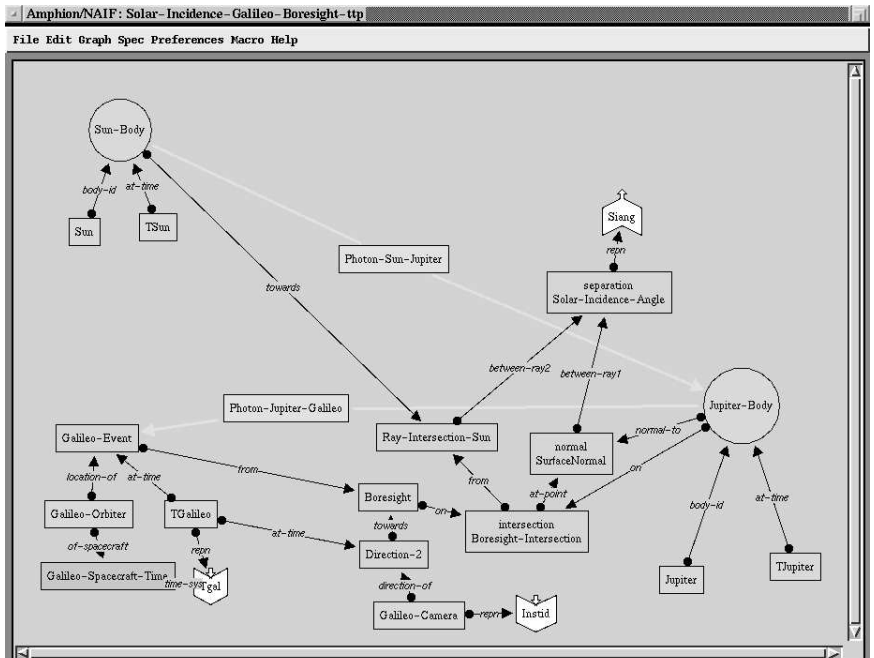
Here we assume that the properties of the components are already known (or supplied by the user), and the circuit specified via a GUI.

Generating a Configuration

The basic operations are

- adding objects;
- deleting objects;
- moving edges between objects to define relationships;
- merging objects (to allow compound operations);
- declaring as input/output.

Slide 18: example



Summary

Amphion provides:

- Interface to library of routines;
- help in forming specification
- automated synthesis of combinations of the routines.