Position Paper:

The Iteration Mechanism in the Inscape Environment

Dewayne E. Perry
AT&T Bell Laboratories, 3D-454
600 Mountain Ave
Murray Hill, NJ 07974

The Inscape Environment

The focus of my research in the Inscape Environment [Perry 85a, 85b, 86b] is on the construction and evolution of large programmed systems. There are two key concepts that form the basis for Inscape's construction and evolution process: 1) the constructive use of formal module interface specifications (expressed in the module interface specification language, *Instress*), and 2) the semantic interconnection model [Perry 86a]. By *constructive* I mean that in addition to providing a formal mechanism for describing module interfaces, Instress specifications are used by the environment in constructing and modifying pieces of software. The *Inform* program construction editor is knowledgeable about the interface specifications, the programming language, and the rules of program construction. As a program is constructed, it enforces the consistent use of the interface specifications according to the rules of program construction.

Instress specifications describe, by means of Hoare-style predicates [Hoare 69], the behavior of operations and the properties of, and constraints upon, data defined within the module. (Extensions to Hoare's approach have been made to include first, the notion of *obligations* as a result of an operation as well as postconditions and second, multiple results in order to specify the meaning of exceptional as well as successful results.) These predicates form the basis for semantic interconnections — they are the elements of interconnections. As a program is constructed, the Inform editor automatically manages the details of the interconnections between pieces of the program, thereby actively participating in the program construction process.

The resulting interconnection structure forms the basis for environment-knowledgable change. Because of the fine-grained interconnections (the predicates), Inscape, through the *Infuse* source change management and coordination subsystem [Perry 86c], is able to determine the implications and the extent of changes and quide the programmer in order to guarantee the completeness and consistency of those changes. It is this part of the environment that provides the actual mechanism for iteration during the development and evolution process.

Sources of Iteration

In general, there are two sources of iteration in the life of a programmed system: the first source is the necessity of making changes; the second source is the effect of any given change. The former may be considered as global iteration and the latter as local iteration. Global iterations occur for a variety of reasons: changes to fix implementation errors, changes to fix design and requirement errors, enhancements

to extend current facilities, and enhancements to introduce new facilities. Local iterations, as a result of a given change, occur because of the implications of changes and the problem of changes cascading — a problem that becomes particularly complex in the context of multiple, concurrent changes as often occur in the development and evolution of large systems.

There is an additional source of iteration that is more or less endemic to the approach used in Inscape: the problem of getting the right abstractions incorporated into a module and developing the appropriate interface specifications. This form of design iteration is common in all systems, but is somewhat hidden in informal development environments. In Inscape, however, it is prominently evident and manifested in the module interface specifications. It is for this reason that particular attention is being given to the problems of incorporating the management of the change process into the support environment and providing automated assistence by a change-knowledgeable environment. Thus, the Inscape environment provides the Infuse subsystem to make it easy to change specifications and determine their implications. A very useful benefit of this provision is that the structure needed for Infuse provides the structure for the two types of iteration mentioned above.

Inscape's Iteration Mechanism: Infuse

The Infuse facilities for managing and coordinating source changes consist of the following parts: a hierarchy of experimental databases; a means of automatically partitioning the components to be changed into experimental databases and of merging the changed components back into the parent database; the notion of change propagation and consistency checking within an experimental database; and private and cooperative workspaces in which to simulate the effects of changes.

The scenario for a set of changes is as follows. The set of modules to be changed is first colected into an experimental database (EDB) and then partitioned recursively into a set of hierarchical experimental databases (HEDBs) where the leaves of the tree are singleton EDBs which contain a single module where the actual changes occur. When all the children EDBs have been deposited into the parent EDB, the changes are propagated to the appropriate components in that EDB and local inconsistencies are reported. Conflicting changes are then resolved and the EDB is repartitioned recursively in order to make the desired changes. When all conflicts and local inconsistencies have been resolved, that EDB is deposited into its parent EDB. The process continues until all the changes have been propagated and all the inconsistencies have been resolved.

HEDBs, then, provide a structure for enforced cooperation, a forum for change negotiation, and a means of managing iterations in the change process. By partitioning the components into separate EDBs recursively, we restrict the bounds of change propagation and consistency checking as well as the number of potential conflicts to be negotiated at any given time.

The optimal oracle for partitioning would be one that answers the question *which pieces will change and how?* Unfortunately, this oracle is not available. An approximation to this oracle is determined by the strength of dependency interconnections (which in Inscape are determined at the semantic level in addition to the syntactic and unit levels). The primary cost to be considered is that of consistency checking and change propagation; secondary costs occur in the creation and merging of EDBs. Thus, it would seem that inconsistencies are more expensive at the upper levels of the hierarchy and less expensive near the leaves.

Partitioning, then, places the heaviest interconnections near the leaves in order to minimize the cost of inconsistencies and to reduce the depth of local iterations.

Because each EDB (except the topmost EDB) has an incomplete set of modules, we need the notion of local consistency in order to determine the implications of changes, deletions, additions, and rearrangements of source code. Basically, this amounts to checking the consistency of those elements that are both defined and used within the EDB. At each level, it is only necessary to check those elements that have not already been found to be consistent.

Workspaces are a complimentary facility to EDBs and provide both cooperative and private means of determining the effects of changes independent of the hierarchy of EDBs, that is, independent of the enforced partitioning. The change simulation mechanism enables the programmer to determine the effects of a change without committing the change. In a cooperative workspace (where a set of programmers together collects their respective modules into the workspace), programmers can determine the mutual effects of a set of changes. In a private workspace (where a programmer selects a set of modules for his workspace independent of those responsible for them), a programmer can determine the effects of changes in his or her module without actually affecting the implicated modules in the workspace.

Summary

Infuse provides the mechanism for managing the process of iterating over source changes. The initial extent of the iteration is approximated by the partitioning algorithm on the basis of weighted interconnections; the extent of subsequent iterations is determined by the inconsistencies and conflicts that occur at each level in the hierarchy.

References

[Hoare 69] C. A. R. Hoare. *An Axiomatic Approach to Computer Programming*. **CACM** 12:10 (October 1969). pp. 576-580, 583.

[Perry 85a] Dewayne E. Perry. *Program Construction and Evolution based on Interface Specifications: Motivation and Overview*. Technical Report. Computer Technology Research Lab, AT&T Bell Laboratories, May 1985.

[Perry 85b] Dewayne E. Perry. *Position Paper: The Constructive Use of Module Interface Specifications*. **Third International Workshop on Software Specification and Design**. IEEE Computer Society, August 26-27, 1985, London, England.

[Perry 86a] Dewayne E. Perry. *Software Interconnection Models. Draft Summary*. Technical Report. Computer Technology Research Lab, AT&T Bell Laboratories, January 1986.

[Perry 86b] Dewayne E. Perry. *The Inscape Program Construction and Evolution Environment. Extended Abstract*, April 1986. Submitted to Practical Software Development Environments Conference, Palo Alto, CA, December 1986.

[Perry 86c] Dewayne E. Perry and Gail E. Kaiser. *Automatically Managing and Coordinating Source Changes in Large Systems. Extended Abstract*. April 1986. Submitted to Practical Software Development Environments Conference, Palo Alto, CA, December 1986.