Introduction

- Aspects of Programming-In-The-Large
  - describing component interfaces
  - managing interface and implementation variants
  - configuring systems from components
  - generating systems from configurations
- Will consider the implications that Inscape’s interface specifications have for version management, system configuration and system generation.

Why Is There A Problem?
- corrections
- improvements
- alternative implementations
- divergent functionality
- different configurations
- Result: a forest of versions

Versions

- Kinds of Versions
  - Successive
    - corrections
    - improvements
  - Parallel
    - alternative implementations
    - divergent functionality
  - Composed
    - different configurations

Questions and Issues

- Important Questions
  - How does one determine the correct version to use?
  - When does a version become a different version?
    - When does a successive version become a parallel version?
    - When does a parallel version become a different version altogether?
  - How will different versions interact?

- Important Issues
  - Identity
  - Equivalence
  - Compatibility
  - Consistency
    - syntactic
    - semantic
### Versions Control Systems

#### Current Mechanisms
- No version control
- Basic version control
- Strongly-typed version control

#### No Version Control

- **Kinds of Versions**
  - may get some notion from file system
  - possibly some vague notions of successive and parallel
  - composition is whatever we throw together

- **Issues**
  - identity, equivalence and compatibility determined by fiat
  - consistency determined by compile/link/execute

- **Evaluation**
  - no control
  - system building very error prone
  - very difficult to reconstruct previous versions

#### Basic Version Control

- **Kinds of Versions**
  - successive, parallel, composed
    - successive and parallel distinguished by version identifiers
    - composition: S-lists that are user defined for unit-level
      components with explicit versions specified

- **Issues**
  - workable solution to version identity
  - no system notion of either equivalence or compatibility
  - consistency determined by compile/link/execute

- **Evaluation**
  - no system determined difference between successive and parallel
  - composition only at system level
  - dependencies are implicit and too coarse grained

#### Strongly-Typed Version Control

- **Kinds of Versions**
  - successive, parallel and composed versions are syntactic objects
    - successive versions: successive revisions
    - parallel versions: determined by fiat
    - composed versions:
      - explicit versions of syntactic objects
      - explicit dependencies — can be general, default, or definitive
    - for small objects as well as systems

- **Issues**
  - better solution to version identity (concatenate p-id and s-id)
  - version equivalence defined in terms of syntactic equivalence
  - notion of compatibility except what is not equivalent is incompatible
  - syntactic consistency guaranteed; semantic consistency by execution

- **Evaluation**
  - on the one hand, notion of equivalence is too broad: allows versions that really are not equivalent
  - on the other hand, notion of equivalence is too narrow: rules out cases that are equivalent
  - notion of compatibility is too strict: bound to name and parameter list for operations
    - e.g., P1(int) is not equivalent to P2(int, bool) and hence is incompatible with P2, even if, in an intuitive sense, P2 is upwardly compatible with P1
Inscape’s Invariant

→ Versions in Invariant
% successive versions: revisions
% parallel versions: environmentally determinable to the extent of different behavior
% composed versions: similar to strongly-typed

→ Issues in Invariant
% Version identity — similar to strongly-typed
% Version equivalence — determinable from the interface specifications
% Version compatibility
  > strict
  > upward
  > implementation
  > system
% Version consistency
  > syntactic: guaranteed (in a looser sense)
  > semantic: guaranteed (within the limits of consistency checking)

Identity and Equivalence

→ Interface Equivalence
% Interface Identity
  I2 is identical with I1 iff
  PRE(I1) = PRE(I2) and
  POST(I1) = POST(I2) and
  OBL(I1) = OBL(I2)

→ Version Identity
  V2 is identical with V1 iff
  the interface of V2 is identical with V1 and
  their implementations are identical

→ Version Equivalence
  V2 is equivalent to V1 iff
  their interfaces are identical

Upward Compatibility

→ V2 is a strictly compatible version of V1 iff
  PRE(V1) \subseteq PRE(V2) and
  POST(V1) \subseteq POST(V2) and
  OBL(V1) = OBL(V2).
  i.e., V2 requires no more than, guarantees no less than, and obligates equally to V1 — captures substitutability

→ V2 is an upwardly compatible version of V1 iff
  PRE(V1) \subseteq PRE(V2) and
  POST(V1) \subseteq POST(V2) and
  OBL(V1) \subseteq OBL(V2).
  i.e., V2 preserves the functionality of V1 while extending it

Implementation Compatibility

→ Exact implementation compatibility
  V2 is exactly implementation compatible with V1 if and only if
  PI(\ldots, V1, \ldots) = PI(\ldots, V2, \ldots)
  i.e., there is no effect on the propagated interface (PI) of the implementation

→ Strong implementation compatibility
  V2 is strongly implementation compatible with V1 if and only if
  PI(\ldots, V2, \ldots) is a strictly compatible version of PI(\ldots, V1, \ldots)
  i.e., there is an effect on the propagated interface, but it is an acceptable one

→ Weak implementation compatibility
  V2 is weakly implementation compatible with V1 if and only if
  the effect of V2 is eventually acceptable
  i.e., eventually the ripples subside
Inscape's Invariant

- **System Compatibility**
  - A version V2 is a system compatible with V1 if and only if V2 is an α implementation compatible version of V1 for all occurrences of V1 in the system, where α is either "exactly", "strongly", or "weakly".

- **Summary of Invariant**
  - A better understanding about the nature of parallel versions
  - A more liberal, flexible method of composition
  - Static determination of syntactic and semantic consistency of composed versions
  - An intuitive definition of version equivalence
  - Intuitive notions of version and system compatibility
  - A unique notion of plug-compatibility

Inquire

Predicate Based Use and Reuse

Introduction

- **Fundamental aspects of use and reuse:**
  - Conceptualization
  - Retrieval
  - Selection
  - Use

- **Compounding considerations:**
  - Multiple levels
  - Various granularities

Inscape — a Specification Based SDE

- **Purpose of Specifications**
  - Express intent of designer
  - Basis of semantic interconnections
  - Basis of conceptualization and retrieval

- **Use of Semantic Interconnections**
  - Capture intent of implementer
  - Detect semantic errors
  - Synthesize interfaces
  - Determine implications of change
  - Basis for selection and use
Current State of the Art: Browsing

**Browsing — Basic Discovery**
- Follow syntactic dependencies
- Use analogical clues for use/reuse
- Static: Masterscope, Cscope, CIA
- Dynamic: Eureka, SDA
- Both: Mview

**Browsing — Drawbacks**
- Get what, not why of dependencies
- The larger the system, the more random the probing

*Useful primarily for building an understanding; Little to offer for retrieval, selection, use.*

Current State of the Art: Retrieval

**Various Approaches:**
- Semantic clues in names
- Keyword schemes:
  - depend on appropriateness
  - keywords = basic concepts
  - BUT, no relationships among concepts
- Prieto-Diaz’ faceted classification
  - conceptual graph
  - weighted terms
- LaSSIE
  - KR for conceptual relationships
  - tractable structure
  - navigation — clarification
  - BUT, handcrafted

*General Problem: independent of system and subject to update problems and conceptual drift*

The Shape of Inquire

**Different Approach to Conceptualization and Naming**
- Concepts represented by predicates
  - Relationships expressed by logical definitions of predicates
  - strength — formal
  - weakness — undecidable
- SDE-managed connection between concepts and source
  - propagate changes in conceptualization to appropriate place
  - changes in behavior are reflected as changes in conceptualization

**Formal Interface Specifications**
- Medium for conceptualization
- Basis for selection, retrieval and correct use
Inquire — Syntactic Browsing

- Merely the traversal of the underlying extended symbol tables and inter/intra module links
- Standard syntactic objects and predicates
  - data flow
  - function flow
  - behavior flow
- 4 basic syntactic browsing commands
  - FIND-DEFS
  - FIND-USES
  - SHOW
  - VISIT
- Basic support for "coarse grained" discovery
- Advantage: added predicate dimension

Inquire — Predicate Based Retrieval

- Context:
  - objects — properties
  - operations — preconditions, postconditions, obligations
  - implementations — state and propagated interfaces
- Challenge:
  - find arbitrary behavior or properties
  - satisfy precondition ceilings and obligation floors
- Usable by both programmer and SDE
- Both single and multiple object retrieval
- Advantages:
  - deductive retrieval
  - direct connection between concepts and objects
  - concepts and objects evolve together

Retrieval — Operations

- Query and Retrieval Commands
  - OPERATION-QUERY-START
  - SHOW-OPERATIONS
  - SHOW_OPERATION-SETS
- Ordering Results for Selection
  - MIN-OPERATIONS
  - MIN-PREDICATES
  - MIN-IMPORTS
  - MIN-EXTRANEOUS
  - MIN-CHANGES

Simplified View of Deductive Retrieval

\[
\begin{align*}
P : & Q, \quad 01: Q \\
R : & X \times T, \quad 02: R \\
U \times T & : V, \quad 03: U
\end{align*}
\]

- QUERY-START
  - prec: <none>
  - post: P
  - obl: <none>
- QUERY-START
  - prec: <none>
  - post: T
  - obl: <none>
- QUERY-START
  - prec: <none>
  - post: V
  - obl: <none>
- SHOW-OPERATIONS
  - prec: <none>
  - post: <none>
  - obl: <none>
- SHOW-OPERATION-SETS
  - prec: [02, 03]
  - post: <none>
  - obl: <none>
Summary

- **Current Use/Reuse Emphasis:**
  - Efficient retrieval at the expense of conceptualization
  - Independent of system structure — update problem

- **Inquire/Inscape — A Different Approach**
  - Specifications are the medium for conceptualization
  - Direct connection that is managed by the SDE
  - Co-Evolution
  - Retrieval, selection, correct use dependent on conceptualization
  - Coarse grained discovery in browsing via predicates