Data Representation In GSRC Bookshelf

Igor Markov

November 20, 1999

Outline

- Basics of data representation, examples
- Bookshelf data formats: motivation and main goals
- Which formats to use: new or existing?
- Cross-learning and reuse
- Grammar tricks and trade-offs
- XML - one large grammar trick (not a magic bullet)
- Conclusions and punch-lines

Basics of Data Representation

- *Semantics, abstract syntax and concrete syntax*
- Semantics = meaning/interpretation
  - determines ***what*** information is represented
  - always ***requires an understanding of the domain***
  - a serial representation may allow multiple semantics (none of them may be obvious from the serial rep.)
  - and vice versa

Abstract Syntax

- ***Determines how the information is organized***
- Does not determine semantics
- Is not determined by semantics alone
- May be "abstracted" from the domain-specifics (to a degree)
- Intimately related to in-memory representations
- Paramount for data format reuse
- Poor abstract syntax may complicate parsing of serial reps.
- Can often be determined from a serial representation and semantics ***(but that's not how it should be designed !)***

Concrete Syntax

- ***A specific serial representation (data format)***
- Does not determine semantics
  - ***we need to explain semantics of data formats***
- May not be compatible with an abstract syntax used for in-memory representations (that's bad)
- When designing a data format
  - ***come up with an abstract syntax first***
  - ***check consistency with expected use scenarios and in-memory representations, discuss with collaborators***
  - ***work on concrete syntax***

Examples

- Semantics: *pins on nets*
- Abstract syntax: *a collection of collections*
- Concrete syntax: e.g., in *.netD, DEF, EDIF* etc
- Sample problems
  - with semantics: **no support for *area pins***
  - with abstract syntax: **no support for *pin addressing (!)***
  - with concrete syntax: **no support for *net names (.netD)***

Bookshelf Data Formats: Motivation and Main Goals

- High-quality serial representations and overall data model
- ***Partial*** and ***incremental*** specification
- ***Avoiding arbitrary restrictions*** => increased reuse
- ***Unambiguous specification***, including
  - semantics
  - desired parser behavior (e.g., error diagnostics)
  - use scenarios
- Pursue more fundamental issues first
  - ***reusable*** => ***fundamental***
  - ***topology*** more fundamental than ***geometry*** *(in PD data model)*
  - ***abstract*** does not always imply ***fundamental***

Main Goals(clarifications)

- Unambiguous specification
  - be careful with "specifying by example"
  - "template formats" versus "real grammars"
- Partial specification
  - specify only **"what matters NOW"**
  - not **"what can possibly ever matter"**
  - parsers with *different detail level* (versus *omnipotent*)
  - **wise defaults** and implied attributes

Incremental Specification

- Can specify more details later
  w/o touching earlier specification
- *Not supported by most existing formats (!)*
- ***Implies multi-file specifications***
- Example: TimberWolf placement formats
- Gains
  - flexibility of specification (can mix and match)
  - space savings
- (XML is neutral regarding incremental specification)

Which Formats to Use: New or Existing?

- Up to slot maintainers
  - are there relevant existing formats?
  - do they go well with bookshelf goals/practices?
  - are they of sufficient consistency/quality?
  - can new formats improve?
- Decision requires public discussion
- Avoid multiple formats using the same abstract syntax
  - carefully study formats in other slots (especially *Fundamental*)
  - *actively look for reuse*
  - contact maintainers if you need compatibility changes
- New formats as "*views*"
  - when existing formats imply unacceptable *redundancy* ***or*** *verbosity*
  - when *abstract syntax* used by existing formats is *inconvenient*

Cross-learning and Reuse

- ***Inventing a new data representation is a priori bad***
  - others have to spend time learning
  - incompatible with existing tools
  - possible mistakes (compared to existing debugged representations)
- Understand work of others, try to reuse
  - understanding ***abstract syntax necessary for reuse***
  - understanding ***semantics - not always necessary***
  - it is ok to suggest/request small changes to enable reuse
- Reuse in new data representations
  - new semantics for existing abstract/concrete syntax
  - specify increments to existing representations

Useful Grammar/Syntax Tricks

- ***Sections*** (aka BEGIN/END) ***are good***
  - add explicit structure (help understand with or w/o manual)
  - localize parsing errors
- ***Data/section types are good***
  - add explicit structure
  - allow to catch accidental mistakes hard to recover otherwise allow extensions via new types
- Pre-numbering (i.e., declaring that N records will follow)
  - convenient for very simple parsers and homogeneous formats
  - ***raises questions about parser behavior, error conditions, diags***
  - e.g., what happens if more/fewer records are present?
  - may conflict with order-independence

Analogy with Programming Langs

- ***Structural programming*** (*clearly applicable to **PD data***)
  - "GOTOs" considered harmful
  - program blocks
  - procedures
  - user-defined types
- ***OO programming***
  - learning justified for relatively complex programs
  - more conceptual approach to programming
  - new possibilities for compiler optimization
  - awful abuses possible
- ***"Are we there yet" with, e.g., Physical Design data (?)***
  - hard ***PD*** problems often have simple inputs (e.g., TSP)
  - little need to represent more complicated structures

Grammar/Syntax Trade-offs

- Addressing
  - by name, specification order, etc?
  - if specification order is not part of semantics, avoid deps.
  - addressing by name requires hash tables and brings order-indep.ce
- Required and optional data
  - requiring too much harms reuse
  - [yet] ***optional specifications: a can of worms***
  - does order of optional specifications matter?
  - are all subsets allowed?
  - advice: ***isolate optionals as much as possible***
  - limit structural options
- ***Default values and behaviors: define carefully***
  - a large number of MS bugs are "default behavior bugs"

XML

- ***Enables tremendous parser and utility reuse***
  - universal parsers
  - universal browsers, search utilities, cataloguers
- Formal specifications of data formats (via DTDs)
  - validating parsers
- Enforces section-based data representation
- Enforces data/section typing (to some degree)
- ***Creating XML versions of existing data formats***
  - ***straightforward (for well-defined formats)***
  - easier to update existing parsers than use a universal XML parser
  - others can exploit universal XML parsers
  - ***conducive to a common data model***

Conclusions

- ***Abstract syntax*** is key for data representation reuse
  - and helps reconcile formats with in-memory structures
- Ambitious goals for bookshelf formats
  - uniform look-and-feel, ***reuse***
  - ***incremental specification***
- Creating a new format is *a priori* bad
  - yet often necessary (look for compromises)
  - study work of others
  - actively seek reuse (full or partial)
- Grammar tricks can do wonders
  - XML as one large grammar trick (not a magic bullet)