Domain specific languages

David Brown

BROOKHAVEN NATIONAL LABORATORY

a passion for discovery

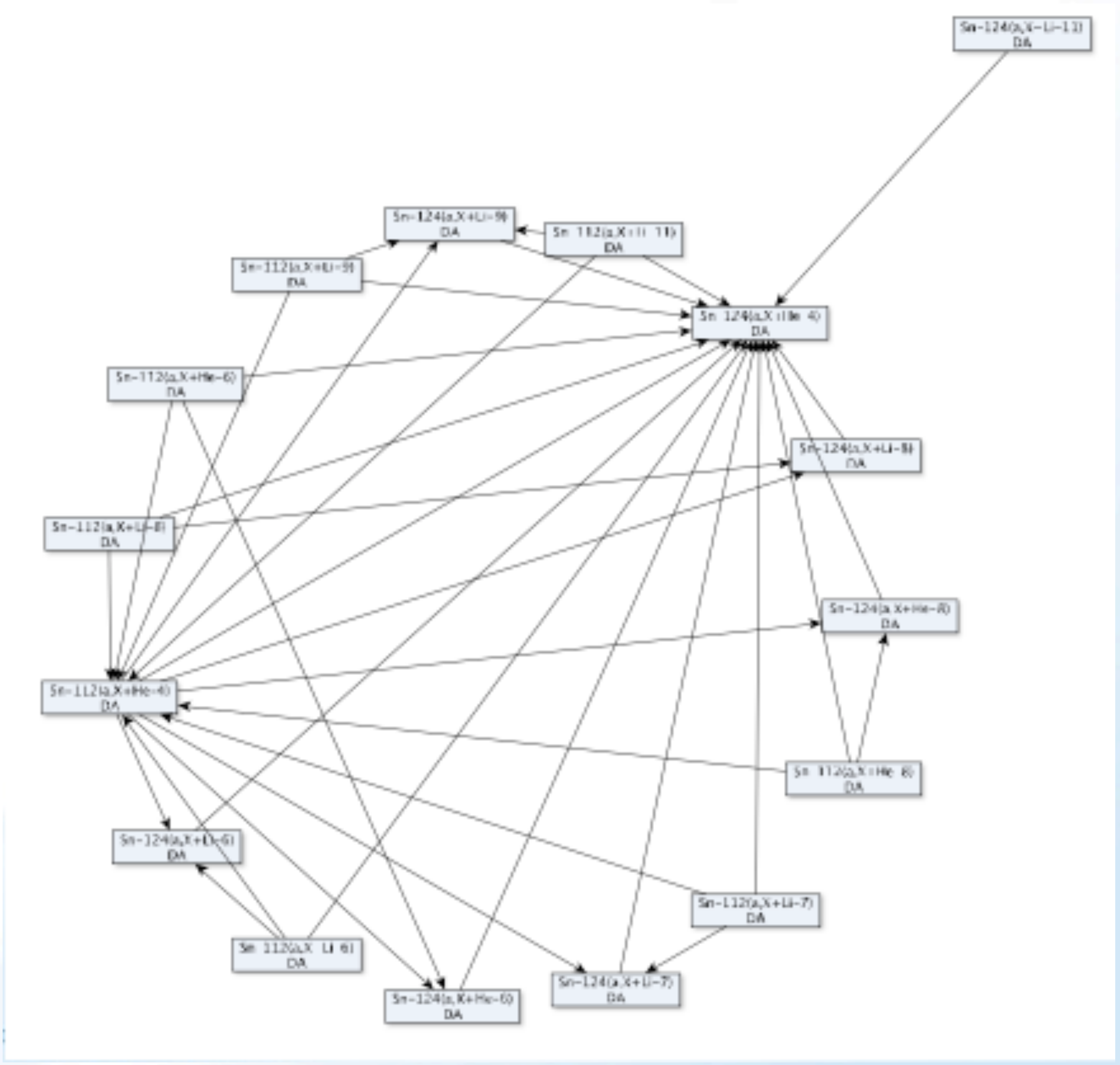
U.S. DEPARTMENT OF ENERGY | Office of Science

Domain specific language vs. explicit xml tagging

- Often encounter a kind of symbol that humans understand, but machines don't: ^{55}Mn(n,2n)
- The logical structure of the symbol could be broken down into xml tagged, machine readable, chunks
  - Humans can no longer read them
  - Logical structure unaltered (obviously)
  - Kind of verbose
- Better is to define a limited set of mini-grammars
  - Would have to encode grammar in processing code
  - Cost commensurate with xml processing of tagged version
  - Still human readable
- Use of mini-grammars should be limited in scope

EXFOR “reaction math” is an example of a well-designed domain specific language

- Mathematical expressions clear and use of parentheses avoids questions about order of operations:
  - e.g. (((reaction string) + (reaction string)) / (reaction string))
- Was able to code grammar in pyparsing in 1 hour (I am slow). Implemented in x4i.
- “Isomer math” breaks this simple grammar



Current GND uses 3 very simple domain specific languages

- Units: SI units are allowed in many places in GND
- Particle and Nucleus designators
- Reaction designators

These or improved versions of these should be part of common subset of formats for ENDF, EXFOR, ENSDF and RIPL

Sample GND code showing units and particle/nuclei designators

```
<particles>
  <particle name="gamma" genre="photon" transportable="true" mass="0 amu"/>
  <particle name="n1" genre="nucleus" transportable="true" mass="1.00866491574 amu"/>
  <particle name="Pu239" genre="nucleus" Jpi="1/2" mass="239.052172899498 amu">
    <level name="Pu239_e0" index="0" energy="0 eV"/>
    <level name="Pu239_e1" index="1" energy="7861 eV">
      <gamma energy="7861 eV" finalLevel="Pu239_e0" probability="1.0"/></level>
    <level name="Pu239_e2" index="2" energy="57276 eV"/>
    ...
    <level name="Pu239_e40" index="40" energy="3.909e6 eV"/>
    <level name="Pu239_c" index="c" energy="u:6.34e5 eV"/></particle>
  <particle name="Pu240" genre="nucleus" mass="240.053813545 amu"/></particles>
```

Elementary particles spelled out, but nuclei have simple SymAA or SymAA_level designation

SI units used in many places to enhance clarity (incl. keV, MeV, etc.)

GND reactions are also clear

- A GND reaction is generally defined by the list of outgoing products:
  - n1 + Pu239 → n1 + Pu239 ← this is MT=2
  - n1 + Pu239 → n1 + Pu239_e1 ← MT=51

- An extra reaction id can also be added, so evaluators can be much more specific:
  - <reaction outputChannel=“n1 + Pu239 [shape elastic]”>…
  - <reaction outputChannel=“n1 + Pu239 [compound elastic]”>…

Advantages & Disadvantages (from Wikipedia)

Some of the advantages:

- Domain-specific languages allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. The idea is domain experts themselves may understand, validate, modify, and often even develop domain-specific language programs. However, this is seldom the case.
- Self-documenting code.
- Domain-specific languages enhance quality, productivity, reliability, maintainability, portability and reusability.
- Domain-specific languages allow validation at the domain level. As long as the language constructs are safe any sentence written with them can be considered safe.

Some of the disadvantages:

- Cost of learning a new language vs. its limited applicability
- Cost of designing, implementing, and maintaining a domain-specific language as well as the tools required to develop with it
- Finding, setting, and maintaining proper scope.
- Difficulty of balancing trade-offs between domain-specificity and general-purpose programming language constructs.
- Potential loss of processor efficiency compared with hand-coded software.
- Proliferation of similar non-standard domain specific languages, i.e. a DSL used within insurance company A versus a DSL used within insurance company B.
- Non-technical domain experts can find it hard to write or modify DSL programs by themselves.
- Increased difficulty of integrating the DSL with other components of the IT system (as compared to integrating with a general-purpose language).
- Low supply of experts in a particular DSL tends to raise labor costs.
- Harder to find code examples.