The Amos Project

Automated Matching of Open Source Code

Carlo Daffara

Conecta Srl, AMOS project
cdaffara@conecta.it
http://www.amosproject.org

FSR2003, 9/10 December 2003, Soissons

One of the great productivity improvements for software engineering was supposed to come from software reuse. During these years, many techniques have been tried:

- Software components (“software IC”)
- Runtime binary package discovery and integration
- Component infrastructures...
- Experience in software engineering has demonstrated that access to source code can reduce the effort needed for reuse (even supporters of black box environments acknowledge this, see Buch, “A plea for grey box components”)
- Free/Open source gives a natural way to explore code reuse, because it gives the technical and legal possiblity to do so

- The problem: finding suitable components (or at least, a small set of candidate packages)
- It is difficult:
  - To find where to search for packages (especially when looking for different development and user communities)
  - To search through many different packages
  - To extract the best ones for the purpose at hand
  - To estimate integration effort

Tried approaches:

Formal methods:

- They give perfect matches
- only small code fragments are handled
- Formal description is required

“Artificial intelligence”:

- Nearly automated, use comments, function names
- Sometimes unable to extract sensible terms
- Multilanguage projects difficult to handle
- Hit rate and precision not very high

Brute force (“the Google! approach”) :

- Simple, fast, requires limited user intervention
- Cannot handle knowledge on the package
- Confused by synonyms, too many false hits

The AMOS approach:

- Ontology for source package description (adds context for searches, allows exchange of semantic information)
- Descriptive vocabulary (including semantics, synonyms, generalizations)
- Describe packages by adding dictionary terms to the ontology
- Use the engine to perform the searches

The AMOS approach:

An extract from the actual ontology:

```
DictionaryItem.property:  definition          (string):1:1
DictionaryItem.property:  synonyms            (string)
DictionaryItem.property:  generalization      (DictionaryItem)
DictionaryItem.property:  translation         ((string),(string))
...
asset.property:           AssetName           (string):1:1
asset.property:           AssetAuthor         (user):1:
asset.property:           AssetMaintainedBy   (user)
asset.property:           AssetHomepage       (string)
asset.property:           AssetDownloadPage   (string):1:
asset.property:           cost                (string)
asset.property:           securityClassification    (string)
asset.property:           certification       (certification)
asset.property:           packageSignature    (string)
```

The AMOS approach: an example

Suppose we want to search for a mail relay:

in(smtp), out(smtp)

any mail server is capable of that (Sendmail, EXIM, postfix..)
And a mail server that saves to spools:

in(smtp), out(smtp), (out(mbox) or out(maildir))

the engine will find not only the traditional mail server (that usually include a local delivery tool), but also all combinations of smtp capable tools plus fetchmail, etc. We can provide different criteria:

in(smtp), out(smtp), out(sql-server)

and this can give us (for example) the Apache James mail server, plus all combinations of SQL servers capable of JDBC.

The AMOS approach: an example

Searching for a Fortran 95 compiler:

in(f95), out(binary-code-x86)

..but we find nothing; so we allow for one generalization, and find the SGI Pro64 that has the description:

in(f95), in(C), in(C++), out(binary-code-IA64)

with the additional rules, that are part of the ontology:

out(binary-code-IA64) has_generalization out(binary-code)

out(binary-code-x86) has_generalization out(binary-code)

The AMOS approach: an example

We can see some more results searching for an F90 compiler:

in(f90), out(binary-code-x86)

..and we add the additional rule that

in(HPF) has_generalization in(f90)

we can find (among the others) the following assembly:

- ADAPTOR, g77

because: ADAPTOR has in(HPF2), (out(f90) or out(f77))

and g77 has in(f77), (out(object-code-x86) or (out(object-code-....)) (as g77 has many, many out targets...)

- Pro64

The AMOS approach: the advantages

- Allows for targeted, precise searches
- The use of dictionary terms for searches allows for high recall and precision
- The engine understands term generalization and specialization
- The engine allows for multipackage searches, composition of results into “assemblies”
- Search results can be ordered using different heuristics

The AMOS approach: the disadvantages

- Package description and dictionaries must be manually specified: time consuming, complex
- Search is constrained: if a term is not in the dictionary, it will never be found
- The target of the engine is specific: people interested in finding and reusing source code
- It is not a general purpose package search engine

The AMOS approach: the disadvantages

- The effort needed is mainly in selecting and properly describe dictionary terms..
- ..but for any domain of expertise, the number of different concepts/terms is limited, and there are overlaps between domains
- Example: protocols, common algorithms

The AMOS approach: other considerations

- The approach lends itself easily to replication
- It is easy to create vertical repositories, by specializing vocabularies
- Interchange is a matter of joining description, eventually merging the conflicting terms
- The ontology is mostly modeled over the BIDM IEEE 1402.1a standard, plus some NHSE and Grid Object Specification-like extensions
- Ontology, database and vocabulary are GFDL'ed

The AMOS approach: initial results and experiences

- The approach seems capable of handling even complex situations...
- ...but it is very difficult to find the right description level - excessively narrow, excessively wide...
- ... and be uniform in the level, across packages, and across different people
- The approach is different from what people is used to, that is, it is not Google!

The AMOS approach: initial results and experiences

- The “synonyms, generalization” approach works well...
- ..but allowing more than one/two generalizations gives way too much results
- for this reason, we will extend the generalization concept, introducing a “subset/superset” concept
- for example: HPF is a superset of F90
- This greatly enhances the search capability, while introducing little work to the dictionary writer

The AMOS approach: initial results and experiences

- We have done several internal tests using our personnel, both experts and not
- Part of our day-to-day operation is finding the right packages for our customers, that fulfill a certain set of requirements
- right now it is done by hand, using an internal database of packages that we have tried, and individual experience
- AMOS has been tested, and (even with a limited number of packages) considered useful..
- ..but you have to express things in a functional way, and searching requires knowledge of the tool

The AMOS approach: initial results and experiences

- After an initial acclimation period, response from internal users is good...
- ...and the engine is extremely fast
- A simulation using all RPMs from RedHat 7.1 demonstrated that on standard hardware, search times are around 40msec, and memory consumption are reasonable...
- ... and a back-of-envelope calculation lends us to believe that it can grow substantially (there is already a query pre-compiler and optimizer)
- As CIAO prolog can target C, it is even possible to use AMOS as a backend engine for APT-like tools
- The engine does have both an in-memory and an external SQL backend (at the moment through MySQL)
- It also scales quite well

www.amosproject.org

- The project site contains a longer description, the previous presentation, and the technical documents
- The source code is available through anonymous CVS access, and the Ciao Prolog environment can be downloaded directly
- We are preparing a more detailed installation guide, to help those that are not proficient with prolog environments
- The prerelease engine is online, but the dictionary content changes frequently as we experiment with it; a stable version is expected soon
- Thanks to the EU IST for funding, and the partners at UPM

Questions?