HFST:
A new division of labour between software industry and linguists

Kimmo Koskenniemi
University of Helsinki

LT is not exploited in software products

- Very few software products make use of language technology (LT)
- Not because there is little need
- Just because it is complicated to integrate modules for a wide array of languages
- Different pieces of code have to be integrated
- Only Microsoft manages to offer fairly good support for many languages

Linguist are poor software integrators?

- Nobody expects that the linguists could solve this problem
- Linguists make analyzers for their (favourite) language using the tools for which they have support
- Many different tools are used on the whole
- No common market place

Software producers are poor linguists?

- Software producers speak only few languages (often only English)
- They are not familiar with the diversity of languages (types of inflection, compounding, grammar, pragmatics, ...)
- Products start in an English environment, and are only afterwards adjusted for other languages

Division of labour has been difficult

- Interfacing programming code is tedious
- Interfacing alien code is time consuming and risky (it can crash the whole application)
- In order to support many languages, many different subroutines need to be interfaced
- The work of a linguist suits only one (or some) formalism and its implementation

Finite-state transducers (FST)

- FSTs are well-known abstract devices with states and transitions (optionally also weights)
- FSTs read strings and output (possibly zero, one or more) strings
- Xerox and ATT have shown that many aspects of language can be efficiently handled with FSTs
- Humans are poor in writing FSTs – but compilers transform lexicons and rules into FSTs

HFST

- Helsinki Finite-State Transducer technology (HFST) is a part of the FIN-CLARIN project
- HFST produces open source tools and language modules (as FSTs)
- HFST is cooperation between several FST research groups and it integrates work of various parties

HFST team

- Krister Lindén (responsible researcher)
- Anssi Yli-Jyrä and Måns Huldén (post doc)
- Miikka Silfverberg, Tommi Pirinen (PhD students)
- Erik Axelson and Sam Hardwick (programmers)
- Kimmo Koskenniemi (consulting and raising funds)

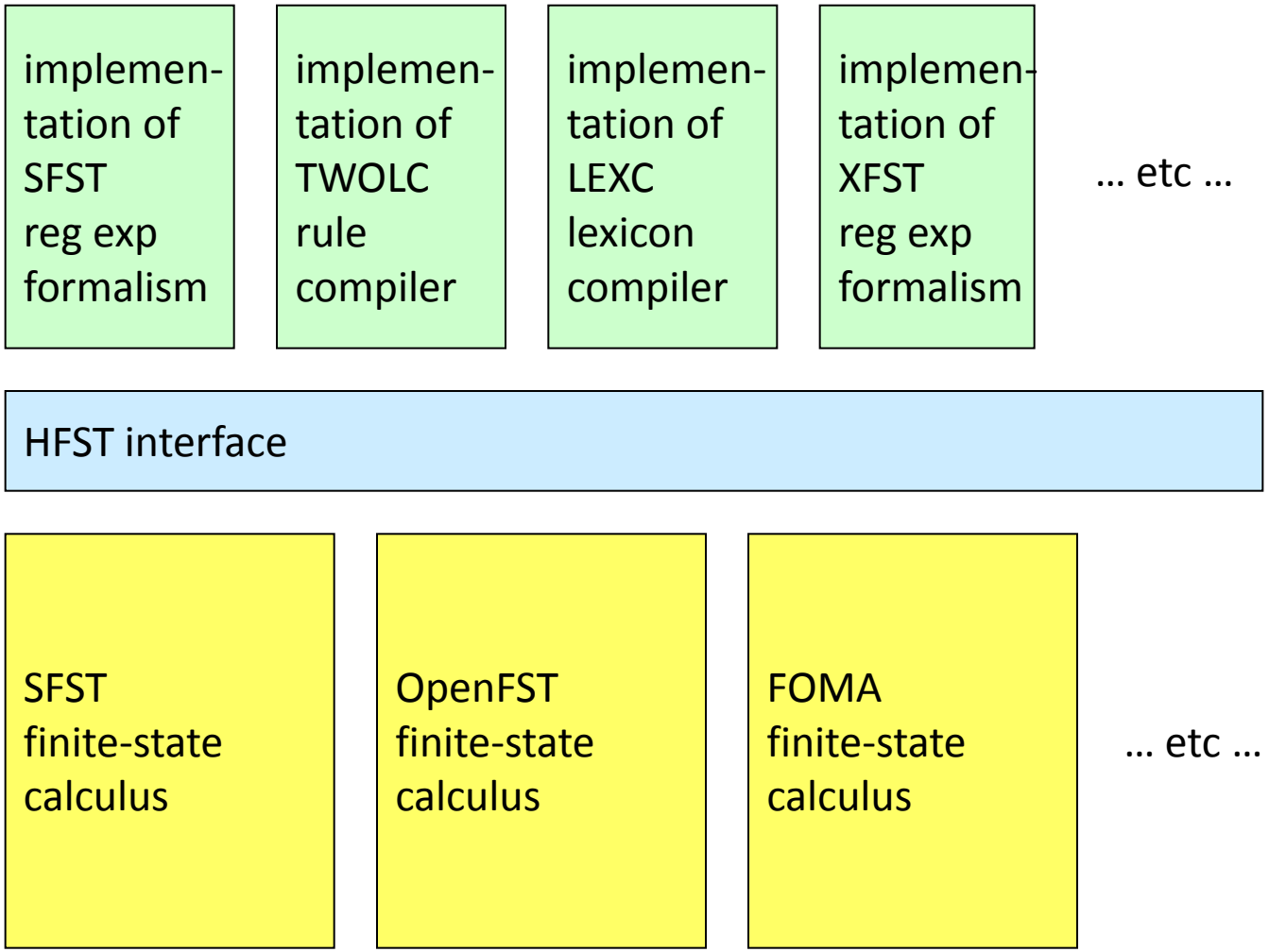
HFST for developers of algorithms

- Dozens of FST software packages have been developed during the last decades, some are no more maintained
- In a package, some algorithms might be good, other ones less optimal
- Some are proprietary, some open source
- Developing a robust FST package is laborious and requires skill and insight

HFST combines some of the best existing FTS software

- SFST by Helmut Schmid (Stuttgart)
- OpenFST (Google research)
- Foma by Måns Huldén (Helsinki)
- Etc. In future
- Packages coexist and can be used through a unified interface in combinations if so desired
- Improved and new algorithms can be developed and added

Design of the HFST



HFST as platform for compilers

- The compiler for a grammar or lexicon formalism can be implemented on top
- The details of individual FST packages are hidden under the HFST interface
- The author of the compiler need not know which underlying package is being used (but may choose individually even single algorithms when needed)

Difficulties in using FST packages directly

- Some packages are good but ...
- Using a package directly is an undoable commitment, no way to change into another
- Each package has idiosyncratic concepts and conventions, many are difficult to detect
- One’s own program starts to reflect these idiosyncrasies and cannot be transferred to another

HFST as platform for lexicons and grammars

- As a proof, a lexicon compiler HFST-LEXC and a two-level rule compiler HFST-TWOLC were made on top of the HFST interface
- Sámi lexicons and two-level grammars (of the Divvun project) were used as a test case
- The SFST and the Xerox regular expression languages can be used for generating all kinds of special applications

HFST for the linguist

- Different styles, cascaded rules and parallel two-level rules are supported and the end result is quite similar FST
- Weighted (statistical) and unweighted (rule-based) descriptions are supported
- Statistical and rule-based models can even be combined
- Morphology and POS tagging now proven

HFST run-time FST

- No matter how the FSTs are compiled, the end result is a compacted fast runtime FST (some 100,000 words/s)
- Long range dependencies are handled with flag diacritics which make some FSTs significantly smaller (at a very slight speed penalty)
- All kinds of linguistic tasks (spelling, hyphenation, search stem generation, ...) are technically similar FSTs which the same (very simple) code runs

HFST run-time code

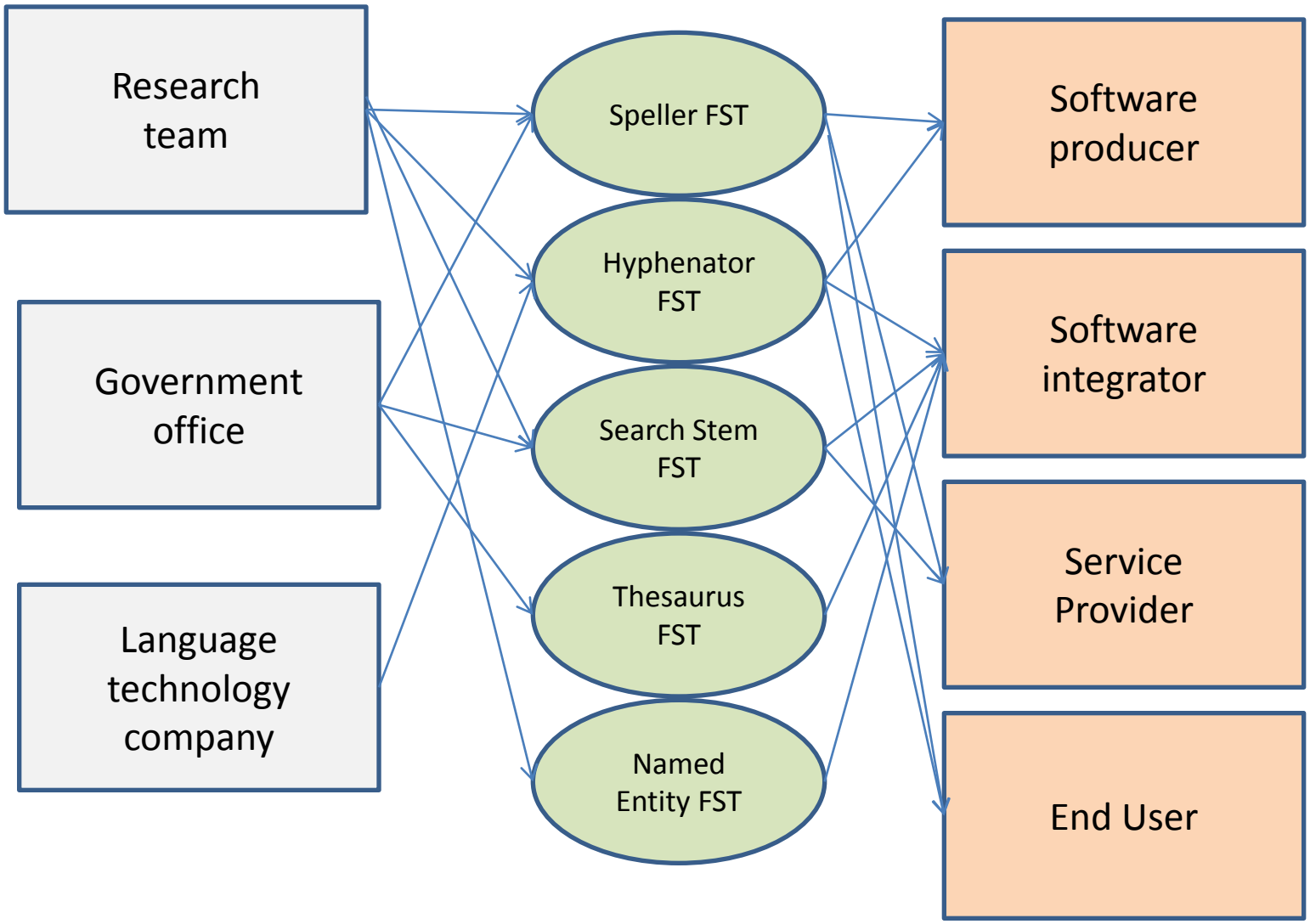
- The code for running the run-time FSTs is short and is provided in several programming languages (C++, Java, Python, …)
- This code is released using the Apache licensese
- Code can be combined with any software (commercial or open source)

HFST through conversion

- In addition to HFST-LEXC and HFST-TWOLC, other modules can be transformed into the these formats and then compiled into FSTs
- Spellers for some 100 languages have been converted in this way into HFST (from Hunspell and other formalisms)
- Conversions from other formats (such as Malaga) would be straight-forward

HFST both for Business and Open Source

- An FST is as proprietary/free as its source
- The tool for creating a proprietary FST may quite well be GNU GPL (no contamination)
- The runtime can be embedded both in commercial and open source software
- Interface to OpenOffice and Mozilla Firefox and Thunderbird has been built


Research team
Government office
Language technology company
Speller FST
Hyphenator FST
Search Stem FST
Thesaurus FST
Named Entity FST
Software producer
Software integrator
Service Provider
End User

Conclusion

- For a class of LT tasks, a common FST format, a supply of tools and a runtime for common programming languages creates a new kind of a market place for
  - LT companies
  - Software producers and integrators
- Peaceful coexistence with open source tools
- Open source modules create a market for higher quality commercial products