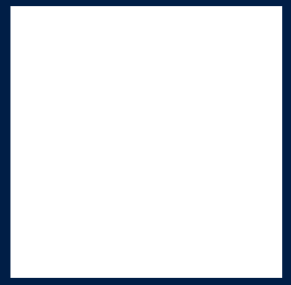
University of Oxford
Department of Computer Science

Saving C02

Top SMW Performance Issues and How to Address Them

Markus Krötzsch
University of Oxford

SMWCon 2011 Fall, Berlin



September 22, 2011

SMW and Performance

- Why care?
  - Critical: “Our site is down again!”
  - Severe: “Our users are complaining about speed.”
  - Tautological: “Our site could speed up a bit.”
  - Economical: “We might use less hardware/energy.”
- “SMW slows down our servers?”
  - In practice: rarely directly
  - But it suggests usage patterns that stress MediaWiki

Optimising SMW
Step 0: Configure MediaWiki for Performance

Configuring MediaWiki for Performance

- Most work is still done by MediaWiki
- Many performance hints available online,
  often very easy to enable
- Do not start to optimise SMW performance before configuring MW properly!

Caching, caching, caching …

- Don't have your server do repeated work!
- **Important caches:**
  - PHP bytecode caches (APC, PHP accelerator, …)
  - Object caches (memcached, ...)
    - Parser cache
    - Message cache
  - Page cache
    - File caches
    - HTTP-level caches (squid, varnish, Web server)

PHP Bytecode Caching

- Cache the “pre-compiled” PHP source code instead of parsing PHP text files on each request
- Possibly the most important cache for small sites, especially if not much editing happens
- Trivial to enable:
  - Install APC for your PHP distribution, e.g. in Debian: “apt-get install php-apc”
  - Done. MediaWiki is much faster now.

Object Caching

- Cache frequently used PHP data objects

- Speeding up user interaction, esp. for logged in users.
- Taking load from the DB.

- Trivial to enable:
  - Enable APC (or some other PHP bytecode cache), then add to LocalSettings.php:

    $wgMainCacheType = CACHE_ACCEL;
  - Sites with many app servers need *memcached*
  - Sites on untrusted servers should avoid this

Parser Caching

- Special object cache for parser output
- Speed up content page rendering if page is unchanged
- Enabled by default
  - *Small* sites could set in LocalSettings.php:

    ```
    $wgParserCacheExpireTime = 2592000; // 30 days
    $wgParserCacheType = CACHE_DB; // for relatively low traffic
    ```

    → assumes relatively low traffic; else keep default
  - Cache will be cleared on edits,
    but not on indirect changes (e.g. SMW #ask)

Message Caching

- Special caches for interface messages
- Speed up all page rendering
- Setup:
  - Basic object caching enabled by default
  - Advanced file based caching since MW 1.16:

    ```
    $wgCacheDirectory = 'PHP-writable, not Web-readable directory';
    ```

    → Directory may need to be created manually

File Caching

- Cache for complete HTML output pages

- Speed up all anonymous viewing
- Simplest page caching method for single servers

- Setup is easy:
  - In LocalSettings.php:

```
$wgUseFileCache = true;
$wgFileCacheDirectory = "{$wgCacheDirectory}/html";  // MW 1.16 default
$wgShowIPinHeader = false; // no personalization of pages
$wgUseGzip = true; // compress files
```

  - No expiry! Refresh on edit/move/purge/... as usual.

HTTP-Level Caching

- Cache for complete HTML output pages

- Speed up all anonymous viewing
- Advanced method for larger sites and multiple servers

- Setup
  - Usually requires installation of tools like Squid or Varnish
  - Possibly also supported by Web server (for individual servers)
  - See manual for details …

The Limits of Caching

- Caching cannot solve all performance issues!
- Even cached outputs need to be recreated fast (see also the famous "MJ incident" at Wikipedia)
- Aggressive caching vs. up-to-date results (esp. for extensions like SMW)

→ Fast page creation remains crucial

Other Generic MW Performance Measures

- Some other settings can help:

```
$wgEnableSidebarCache = true;
$wgDisableCounters = true;
$wgMiserMode = true; // cache expensive Specials
$wgCompressRevisions = true; // needs PHP zlib
// For sites with many writes:
$wgAntiLockFlags = ALF_NO_LINK_LOCK | ALF_NO_BLOCK_LOCK;
```

- Use fast native tools for Diff (set $wgDiff and $wgDiff3)
- Make sure PHP mbstring module is installed
- Use low setting for $wgJobRunRate; maybe “0” and use cron to run update jobs regularly (runJobs.php)
- Scale hardware (more/better, split DB/Web/caching)

Optimising SMW
Step 1: Know Your Problem

What's the Problem?

- Difficult to identify cause of performance problems
- Questions to ask:
  - Where does the problem occur?
    (CPU, Mem, DB, …)
  - When does the problem occur?
    (Particular function, high traffic, DoS attack, …)
  - Which software components are involved?
    (MW, SMW, extensions [of SMW], …)

Tracking Down Performance Issues

- Server level: which process does the work?
- Log files:
  - Apache log, also to identify impolite crawlers
  - MySQL log, especially slow query log
- MediaWiki profiling: get times for individual methods
  → requires familiarity with MW code to interpret

Careful: logging/profiling also slows down a site!

Tracking Down Performance Issues

- Know your wiki, know your users
- What are the complex templates/queries on your wiki?
- Which functions are reported to be too slow?
- What are the slow pages?
  - Careful: slowdown can come from background jobs, independent of the page that is displayed

Performance Impact of Semantic MediaWiki

- Possible causes for SMW performance issues:
  - Many small DB reads (e.g. find default Form)
  - Many small DB writes (update data on *all* edits)
  - Complex read queries (difficult #ask queries)
  - Long, complex pages (#ask makes this easy)
  - Complex templates (SMW users *love* templates)
- Two main areas: database activity and PHP activity
- PHP activity dominates on small sites

Optimising SMW
Step 2: Escaping Template Hell

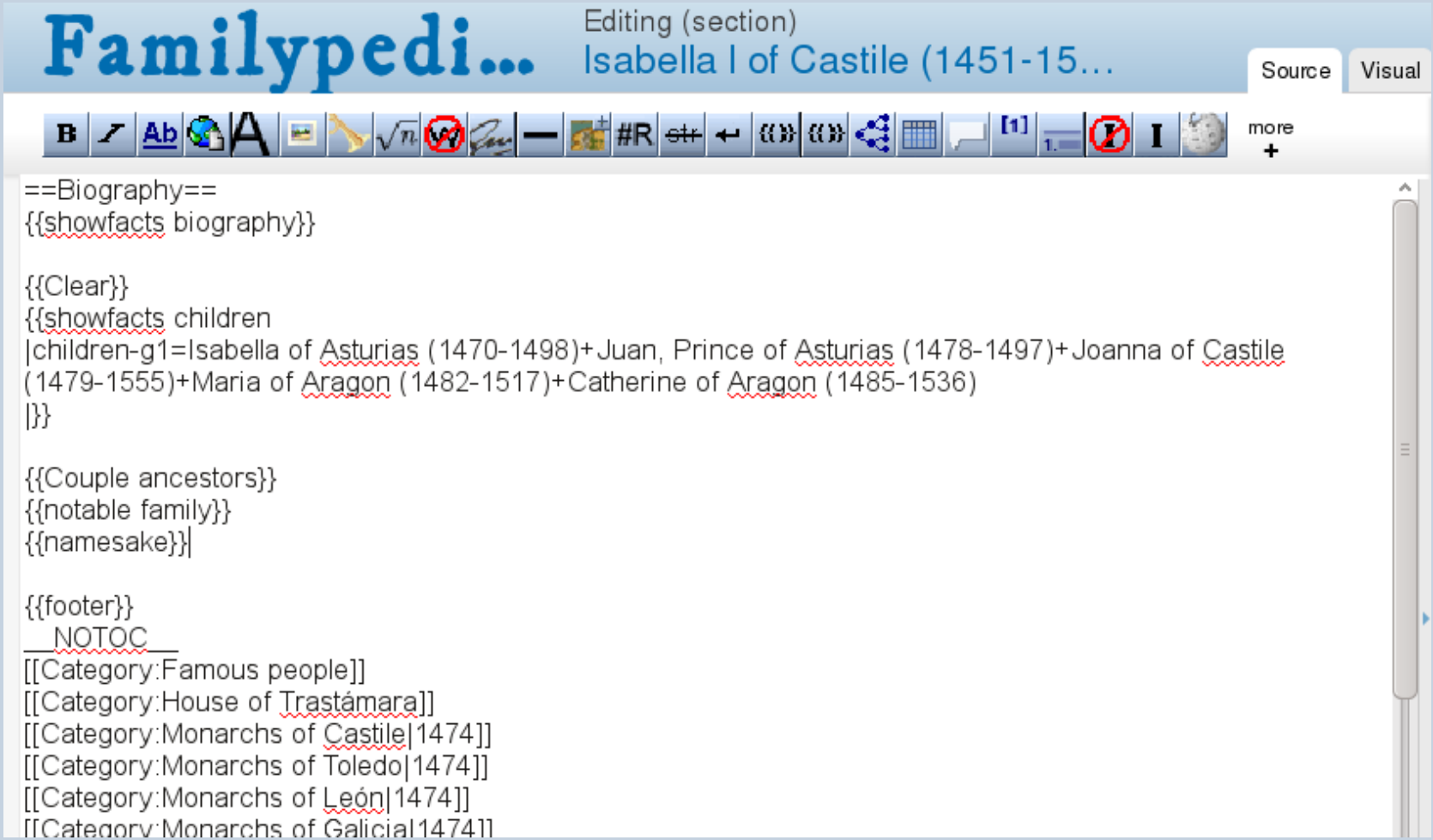
Templates and Parser Functions

- Original intention:
  - Templates: re-use common parts of a page for consistent layout, customised with parameters
  - Parser functions: enable some further customisation, e.g. with conditionals
- What users make of it:
  - In-page data processing in an extremely inefficient and unreadable script language
  - Even more fun when mixed with SMW queries

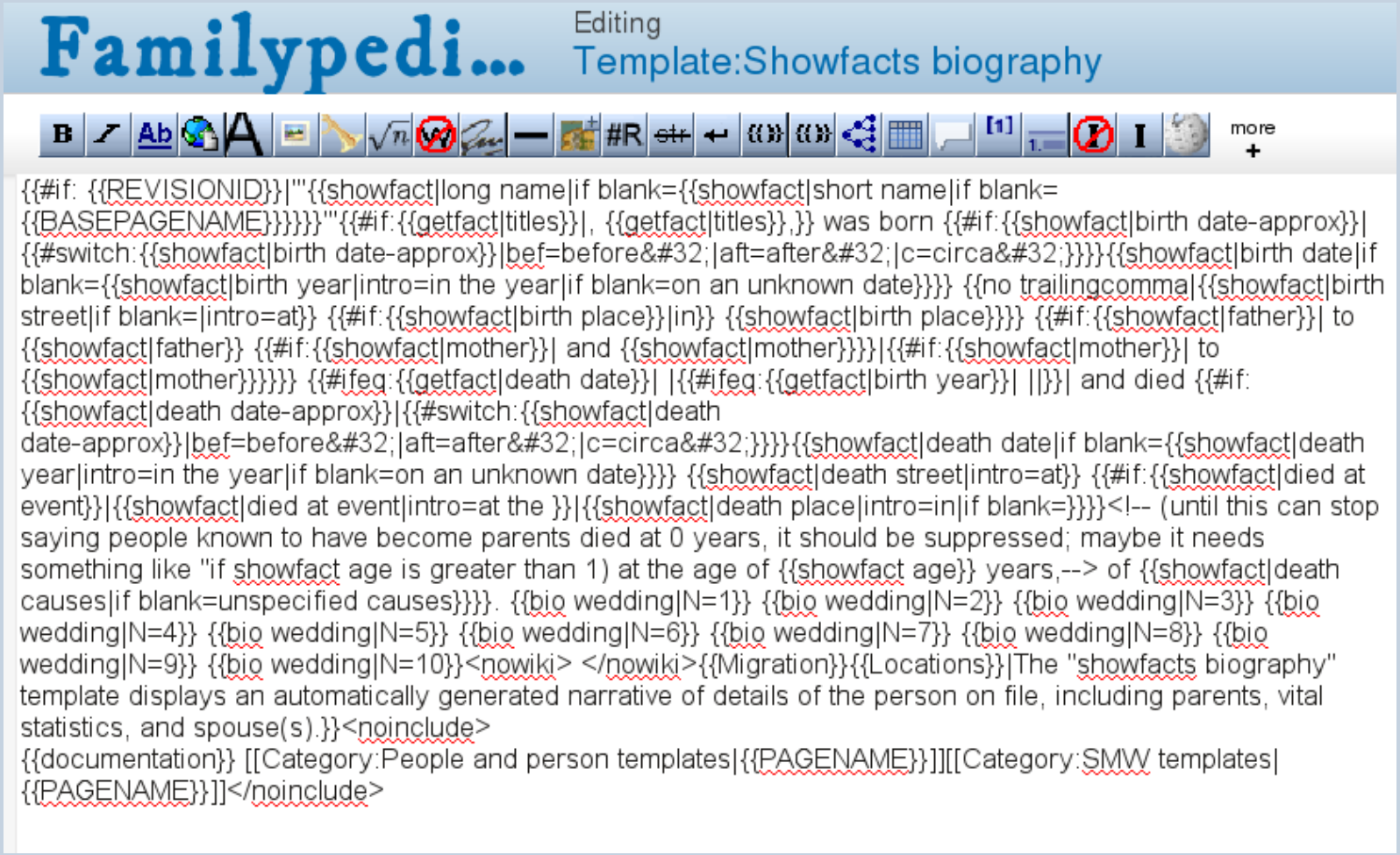
Example


Familypedia
Create new article
Help: Creating articles
Indexes of ancestors
Isabella I of Castile (1451-1504)
0 Talk
Like
Edit with form
Main
tree
descendants
Biography Edit
Isabella I of Castile, Queen regnant of Castile, Queen regnant of Toledo, Queen regnant of León, Queen regnant of Galicia, was born 22 April 1451 to John II of Castile (1405-1454) and Isabella of Portugal (c1428-1496) and died 26 November 1504 of unspecified causes. She married Ferdinand II of Aragon (1452-1516) 19 October 1469 . Ancestors are from Spain, France, Germany, Byzantium, Belarus, United Kingdom, Russia.
Isabella of Castile
Birth: April 22, 1451
Death: November 26, 1504
Father: John II of Castile (1405-1454)
Mother: Isabella of Portugal (c1428-1496)
Spouse: Ferdinand II of

Example


Familypedi...
Editing (section)
Isabella I of Castile (1451-15...
Source
Visual
more
==Biography==
{{showfacts biography}}
{{Clear}}
{{showfacts children
|children-g1=Isabella of Asturias (1470-1498)+Juan, Prince of Asturias (1478-1497)+Joanna of Castile (1479-1555)+Maria of Aragon (1482-1517)+Catherine of Aragon (1485-1536)
|}}
{{Couple ancestors}}
{{notable family}}
{{namesake}}
{{footer}}
__NOTOC__
[[Category:Famous people]]
[[Category:House of Trastámara]]
[[Category:Monarchs of Castile|1474]]
[[Category:Monarchs of Toledo|1474]]
[[Category:Monarchs of León|1474]]

Example


Familypedi...
Editing
Template:Showfacts biography
more
{{#if: {{REVISIONID}}|'''{{showfact|long name|if blank={{showfact|short name|if blank={{BASEPAGENAME}}}}}}'''{{#if:{{getfact|titles}}|, {{getfact|titles}},}} was born {{#if:{{showfact|birth date-approx}}|{{#switch:{{showfact|birth date-approx}}|bef=before |aft=after |c=circa }}}}{{showfact|birth date|if blank={{showfact|birth year|intro=in the year|if blank=on an unknown date}}}} {{no trailingcomma|{{showfact|birth street|if blank=|intro=at}} {{#if:{{showfact|birth place}}|in}} {{showfact|birth place}}}} {{#if:{{showfact|father}}| to {{showfact|father}} {{#if:{{showfact|mother}}| and {{showfact|mother}}}}|{{#if:{{showfact|mother}}| to {{showfact|mother}}}}}} {{#ifeq:{{getfact|death date}}| |{{#ifeq:{{getfact|birth year}}| ||}}| and died {{#if:{{showfact|death date-approx}}|{{#switch:{{showfact|death date-approx}}|bef=before |aft=after |c=circa }}}}{{showfact|death date|if blank={{showfact|death year|intro=in the year|if blank=on an unknown date}}}} {{showfact|death street|intro=at}} {{#if:{{showfact|died at event}}|{{showfact|died at event|intro=at the }}|{{showfact|death place|intro=in|if blank=}}}}<!-- (until this can stop saying people known to have become parents died at 0 years, it should be suppressed; maybe it needs something like "if showfact age is greater than 1) at the age of {{showfact age}} years,--> of {{showfact|death causes|if blank=unspecified causes}}}}. {{bio wedding|N=1}} {{bio wedding|N=2}} {{bio wedding|N=3}} {{bio wedding|N=4}} {{bio wedding|N=5}} {{bio wedding|N=6}} {{bio wedding|N=7}} {{bio wedding|N=8}} {{bio wedding|N=9}} {{bio wedding|N=10}}<nowiki> </nowiki>{{Migration}}{{Locations}}|The "showfacts biography" template displays an automatically generated narrative of details of the person on file, including parents, vital statistics, and spouse(s).}}<noinclude>
{{documentation}} [[Category:People and person templates|{{PAGENAME}}]][[Category:SMW templates|{{PAGENAME}}]]</noinclude>

Example

Familypedi... Editing Template:Showfact


```
<includeonly>{{showfact/aux1|{{#show:{{{page|{{BASEPAGENAME}}}}}}|default=|link={{#switch:
{{{link|}}}|none=none|subject=subject|all}}|intro={{#if:{{{intro|}}}|{{{intro|}}}&#32;}}|outro={{{outro|}}}|?
{{{1}}}}}|page={{{page|{{BASEPAGENAME}}}}}|link={{#switch:{{{link|}}}|none=none|subject=subject|all}}|intro=
{{#if:{{{intro|}}}|{{{intro|}}}&#32;}}|outro={{{outro|}}}|default={{{if blank|}}}|shortname={{{2|}}}|property={{{1}}}
}}</includeonly><noinclude>
{{documentation}}
[[category:facts templates]]</noinclude>
```

Example: Results

- Significant time for building page
  - parsing templates is slow and memory intensive
  - easily above 1 min for page rendering
- Many database reads …

Example: Results

- Significant time for building page
  - parsing templates is slow and memory intensive
  - easily above 1 min for page rendering
- Many database reads …

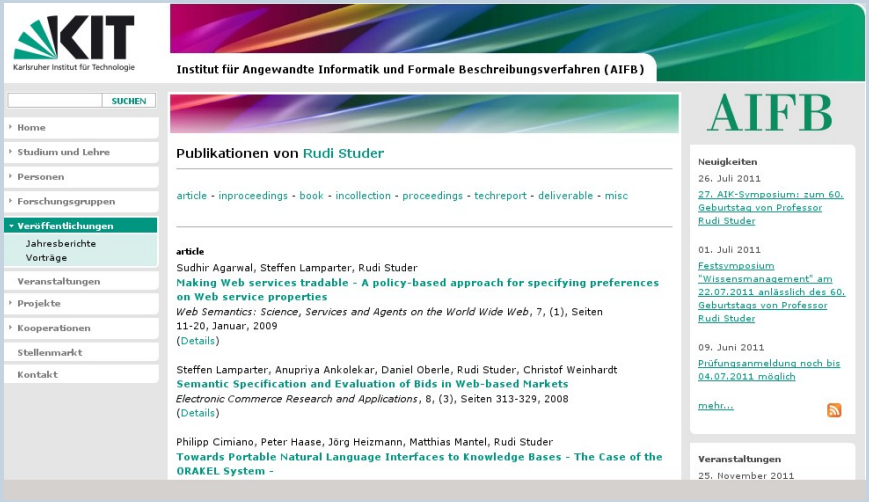
… more than 3000 queries for one page!

Toward a Solution

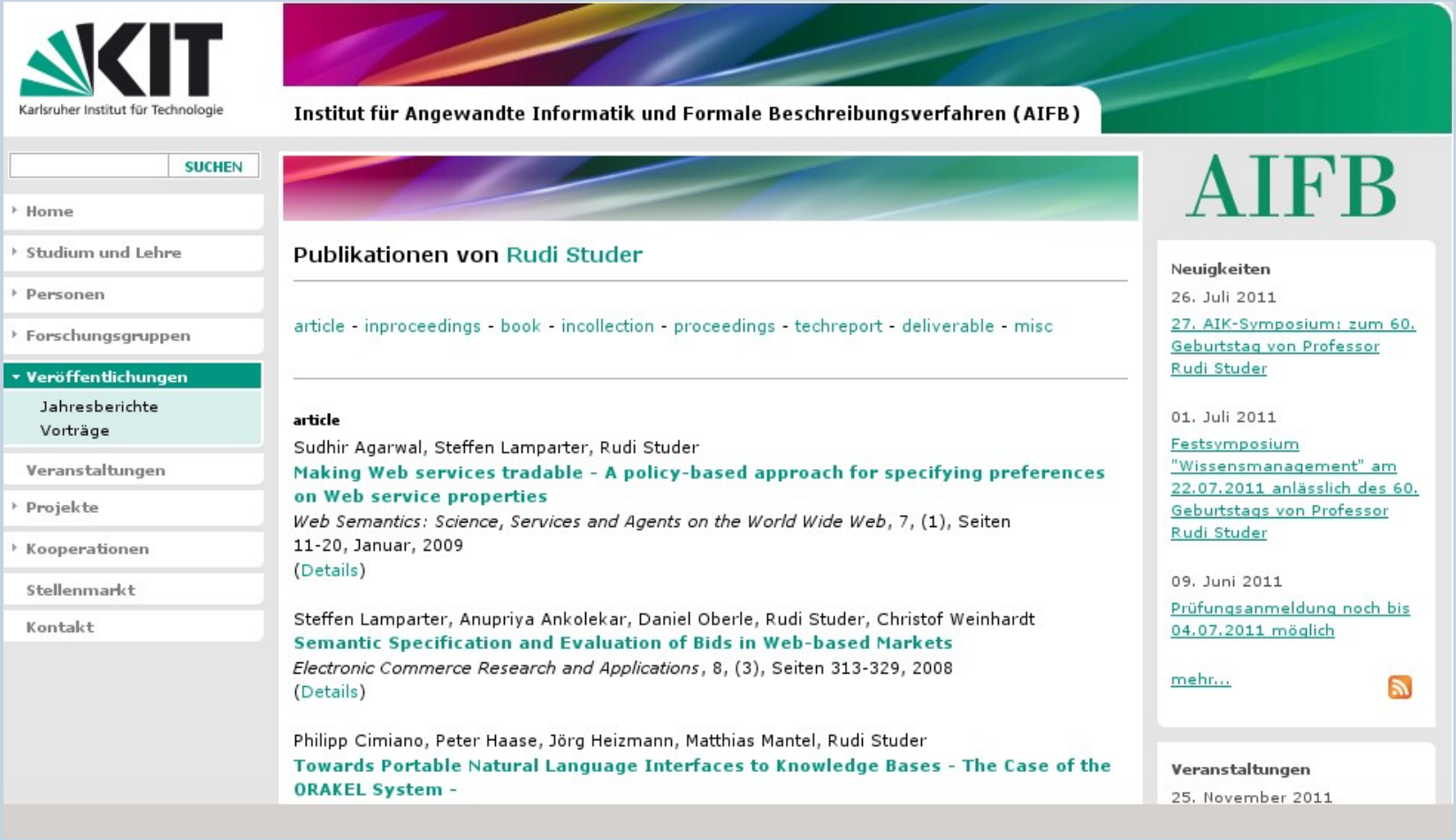
- Are these DB reads really needed?
- Good: Global queries
  - Aggregate data from many pages (“Show all upcoming events,” “How many cities are there?”)
  - Answer global questions (“What's the largest city?”)
- Bad: Local formatting
  - Fetch and process data for a single source (#show)
  - Complex result formatting (format=template)

Eliminating Local Formatting

- Local formatting expensive: complex template parsing, easily for hundreds of query result items
- Example: AIFB Portal publication listing



Eliminating Local Formatting: Example



Eliminating Local Formatting

- Publication data already known when editing source (i.e. publication page)
  → no need to rebuild it on all query pages!

- Solution:
  - Store complete, pre-formatted output string on source page (property of Type:Text)
  - Simply recall this text in all queries
    → Pre-expanded wiki text,
    no templates, no #show, no parser functions

Eliminating Local Formatting: Summary

- A way to escape Template Hell (at least at query time)
- Essentially a user-controlled, DB-based parser cache for snippets used elsewhere
- Some work needed to implement, requires understanding of wiki structure
- Harder to apply if item formatting includes non-local information (stale data problem: even source page refresh will not update the SMW database records!)

Optimising SMW
Step 3: Crazy Queries

SMW Queries

- Three stages:
  - (1) Page selection (query string)
  - (2) Data selection (printout parameters)
  - (3) Formatting
- Local formatting is a combination of (2) and (3): many queries, many templates, many parser functions
- But slow MySQL queries (see log) usually are based on (1)

Crazy Queries

- Example annotation model:

  <Bus line> Property:stops at <Bus stop>
  <Bus stop> Property:served by <Bus line>

- Example query:

  {{#ask: [[served by.stops at::Some bus stop]] }}

Crazy Queries

- Example annotation model:

  <Bus line> Property:stops at <Bus stop>
  <Bus stop> Property:served by <Bus line>

- Example query:

  {{#ask: [[served by.stops at.
  served by.stops at::Some bus stop]] }}

Crazy Queries

- Example annotation model:

  <Bus line> Property:stops at <Bus stop>
  <Bus stop> Property:served by <Bus line>

- Example query:

```
{{#ask:  [[served by.stops at.
          served by.stops at.
          …
          served by.stops at.
          served by.stops at::Some bus stop]] }}
```

Three Possible Solutions

- Stop such madness early on: limit query complexity
- Aggressively cache query results: Concepts
- Use a query engine that can handle this: RDF store

Limitting Query Complexity

- SMW has a number of parameters for this
- Set in LocalSettings.php – defaults:

```
$smwgQMaxSize = 12;
$smwgQMaxDepth = 4;
$smwgQFeatures = SMW_PROPERTY_QUERY | SMW_CATEGORY_QUERY |
                 SMW_CONCEPT_QUERY | SMW_NAMESPACE_QUERY |
                 SMW_CONJUNCTION_QUERY | SMW_DISJUNCTION_QUERY;
$smwgQMaxInlineLimit = 500;
$smwgQPrintoutLimit = 100;
$smwgQDefaultNamespaces = null;
$smwgQComparators = '<|>|!~|!|~|≤|≥|<<|>>';
$smwgIgnoreQueryErrors = true;
```

Concept Caching

- Concepts as “Virtual Categories”:
  SMW page selection queries that can be re-used in #ask
- Managed on pages in Concept: namespace
- May be pre-computed with maintenance script SMW_conceptCache.php
- Rationale: Usually very few queries are really “hard,” concepts allow manual control of what can be supported

Concept Caching: Example configuration

- In LocalSettings.php:

```
// Restict size of inline queries:
$smwgQMaxSize = 6;
$smwgQMaxDepth = 3;

// Allow larger concept queries
$smwgQConceptMaxSize = 20;
$smwgQConceptMaxDepth = 8;

// Hard queries rely on pre-computed cache (default):
$smwgQConceptCaching = CONCEPT_CACHE_HARD;
```

RDF Stores

- SMW can use RDF database systems instead of RDBMS to answer queries
- Special form of “noSQL” databases
- Different architecture, optimised for different query types
- More in tomorrow's talk “Connecting SMW to RDF Databases”

Optimising SMW
Step 4: Improve the Software

Do it Yourself (or find someone who does it for you)

- Instead of using complicated queries and formatting templates, write some PHP code to do it directly
- Help needed in making SMW more efficient
  - For example, to take advantage of object caching
- Some extensions need help to become more efficient
  - For example Semantic Drilldown

Optimising SMW
Conclusions

Summary & Conclusion

- Step 0: Configure MediaWiki properly for your site!
  (At the very least: install APC!)
- Step 1: Find out where your problem is (SMW?)
- Step 2: Escape Template Hell
- Step 3: Restrict or tame Crazy Queries
- Step 4: Develop your own Perfect Solution

Optimisation can be easy. Give it a try.

Further Reading

- http://www.mediawiki.org/wiki/Manual:Performance_tuning
  and the links in “See also” on this page, esp.:
  - http://www.mediawiki.org/wiki/User:Aaron_Schulz/How_to_make_MediaWiki_fast
  - http://www.mediawiki.org/wiki/User:Ilmari_Karonen/Performance_tuning
- http://www.mediawiki.org/wiki/Manual:Cache
- http://www.mediawiki.org/wiki/Manual:Configuration_settings#Cache
- http://semantic-mediawiki.org/wiki/Help:Configuration
  especially the section on Query Settings
- http://semantic-mediawiki.org/wiki/Help:Concept_caching
- http://semantic-mediawiki.org/wiki/Help:Using_SPARQL_and_RDF_stores