The SUSE Linux Feature Tracking Tool

Cornelius Schumacher
Klaas Freitag

Novell

SUSE Feature Management

- Motivation
- Architecture
- Clients
- Demonstration
- Conclusion

Motivation

Motivation

- Managing Features for SUSE Linux Products
- Feature request come in many flavours
  - Feature requests by partners
  - Requirements by product management
  - Enhancement suggestions by users
  - ...
- Several thousand requests
- Development for multiple products in parallel

Involved Persons

- Partners
- Technical Account Managers
- Product Managers
- Project Managers
- Team Leaders
- Developers
- Documentation Writers
- Quality Assurance

Process

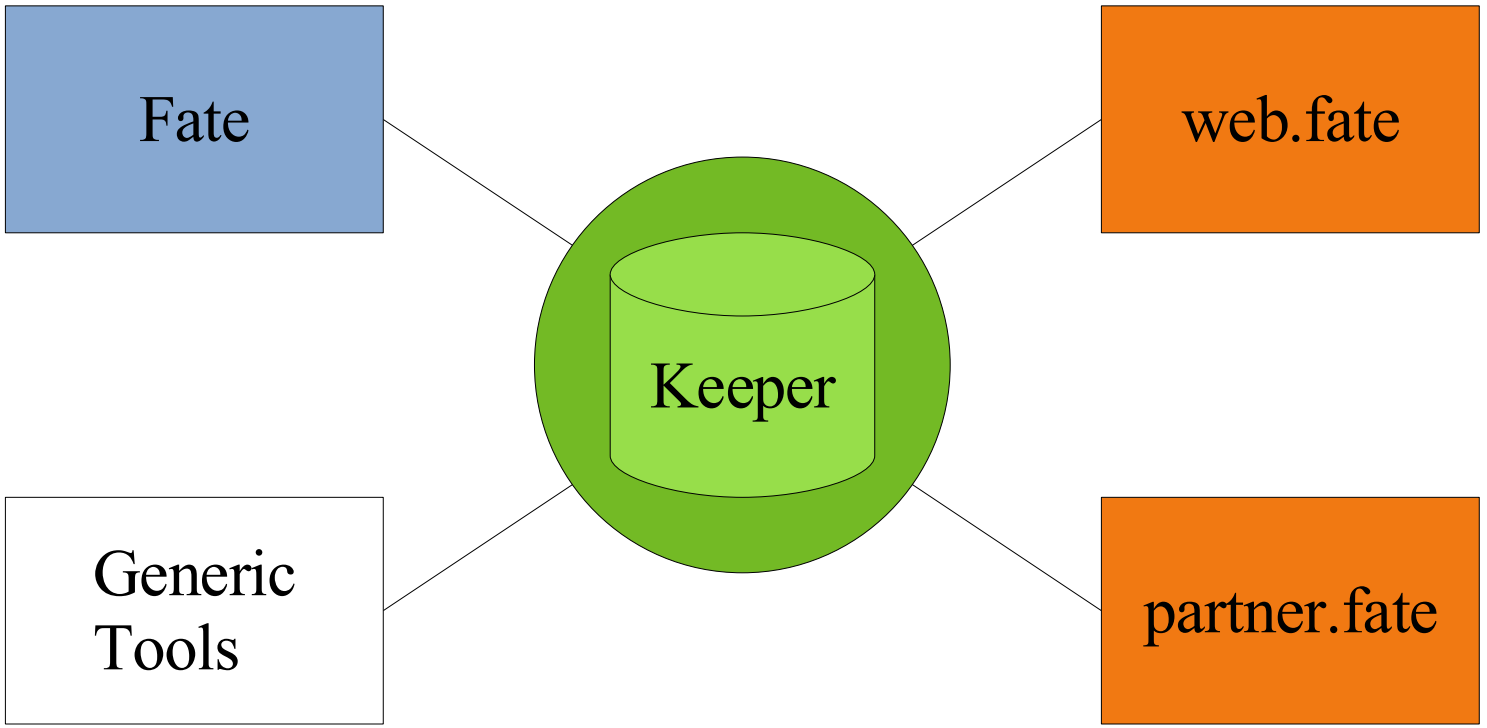
- Request filed by technical account manager on behalf of partner, product manager, engineers
- Review by product and project managers
- Review by team leader
- Implementation
- Test
- Release

Feature Data

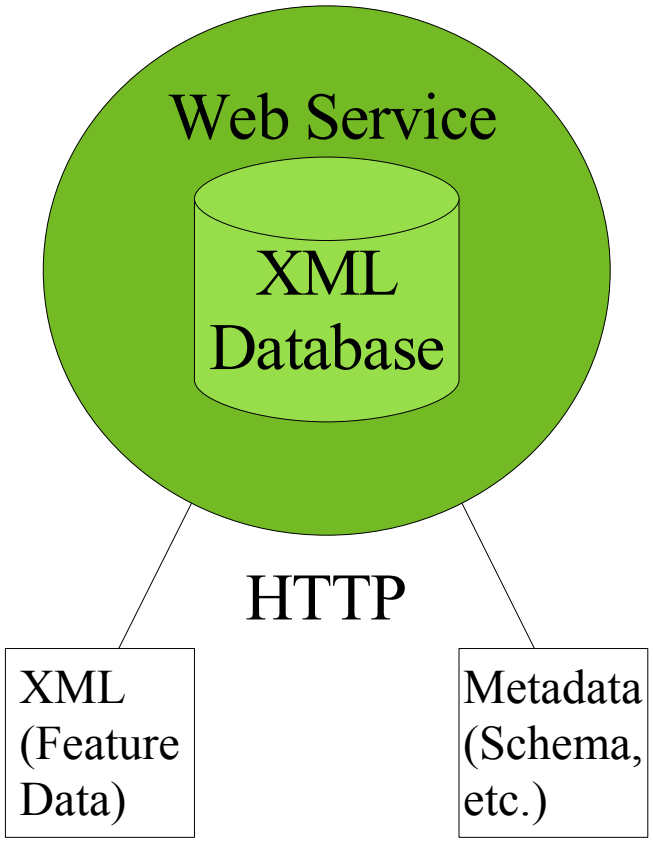
- Id
- Title
- Categories
- Description
- Product context (product information, status, priority)
- Actors (involved persons)
- Partner context (partner information, business case)
- References
- Impact on documentation and QA
- Comments, discussion
- ...

Architecture

SUSE Feature Management System



Keeper



- Data Storage
  - XML Database (Berkeley DB XML)
- Client Interface
  - Web Service (REST, XML over HTTP)
  - Data Access (XML)
  - Schema Access (XML Schema)
  - Metadata Access
- Email change notification
- Administration Interface

Client Access Protocol

- Protocol
  - REST-based web service
  - XML over HTTP
  - XQuery interface
  - Optimistic locking (conflict detection on write)
- Data
  - Feature data (XML)
  - XML Schemas
  - Style Sheets (XSLT) for HTML and text views
  - Metadata (categories, product lists, ...)

Additional Client Access Features

- Versioning
  - Change history
  - Access to previous revisions of documents
- Diffs between revisions
  - XSLT converts XML to text
  - text diff
- Different data representations
  - HTTP Accept headers
  - XML, HTML, Text
- Schema update handling
  - XSLTs for transforming data on schema upgrade

Feature Data Example (1/3)

```
<feature schemarevision="4" id="300666" revision="1" >
    <category>Kernel</category>
    <category>SUSE Linux Enterprise Server</category>
    <title>Exciting Feature</title>
    <description>
        <richtext>
            <p>This is the most exciting feature of the next
            release.</p>
        </richtext>
    </description>
    ...
```


Novell

Feature Data Example (2/3)

```
...
<productcontext>
    <product><productid>SLES-10</productid></product>
    <status><new/></status>
    <priority>
        <mandatory/>
        <owner><role><productmanager/></role></owner>
    </priority>
</productcontext>
...
```

Feature Data Example (3/3)

```
...
<discussion>
  <comment created="2006-06-23T01:36:26" id="1" >
    <author>
      <person><email>cschum@suse.de</email></person>
    </author>
    <richtext>
      <p>Very important, because customers really want it.</p>
    </richtext>
  </comment>
 </discussion>
</feature>
```

Client Access API

- URL: `http://keeper.suse.de/feature/123`
- HTTP Request
  - Header
    - `GET feature/123`
    - `Accept: text/xml`
    - `...`
  - XML Body
    - `<feature>...</feature>`
- HTTP Response with XML Body
    - `<status>...</status>`

Basic Client Access API

- Read all features
  - `GET feature`
- Read features for query
  - `GET feature?query=/feature[category=Kernel]`
- Read one feature by id
  - `GET feature/123`
- Update one feature
  - `PUT feature/123`
- Create new feature
  - `POST feature`

Extended Client Access API

- Metadata
  - `GET schema/feature`
  - `GET resource/feature.xslt`
- Versioning
  - `GET feature/123/history`
  - `GET feature/123/2`
  - `GET feature/123/diff?oldrev=2&newrev=3`

XQuery

- Standard query language for XML data
- Keeper implements full XQuery spec
- Clients mostly use only subset querying feature objects

- Examples:
  - Features with a given category

    ```
    feature[category="Kernel"]
    ```

  - All new features

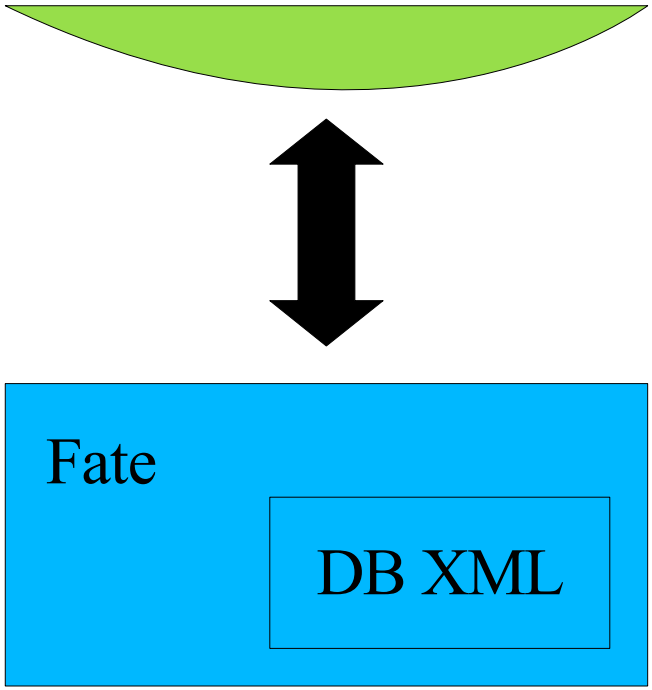
    ```
    feature[productcontext/status/new]
    ```

  - All features with actor “abc” and no priority

    ```
    feature[actor/person/email="abc" and not
     productcontext/product/priority]
    ```

Clients

Fate



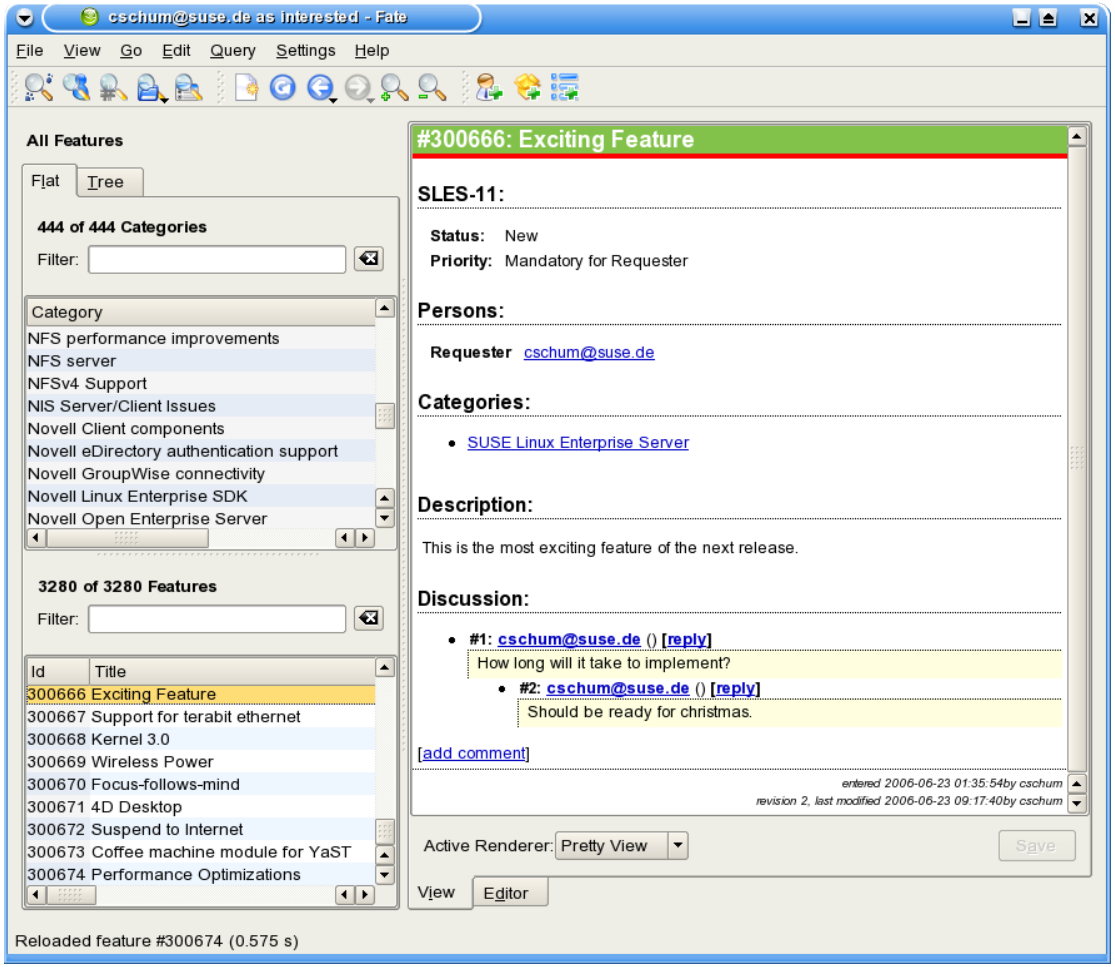
- Rich native GUI client

- Queries
- Browsing
- Feature View
- Feature Editor
- Validation
- Offline Access (work in progress)
- Printing

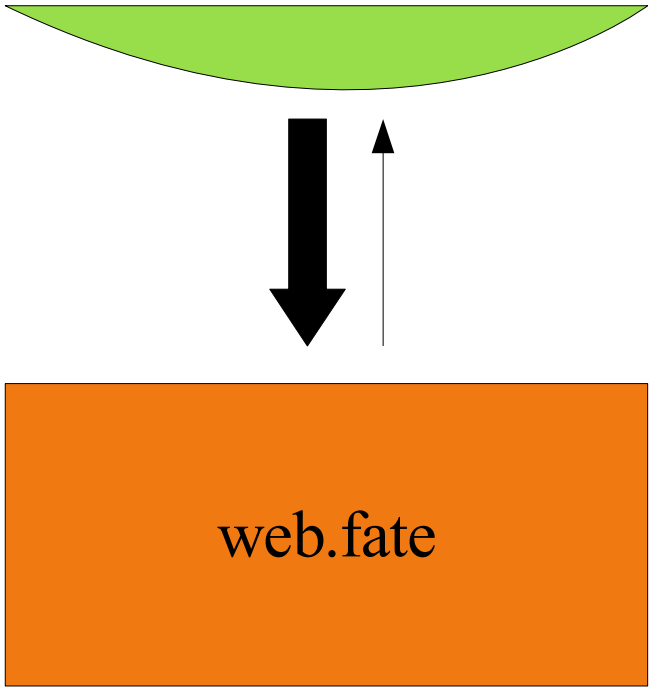
Fate Implementation

- Uses the KDE framework
  - kioslaves for HTTP access
  - KHTML to render HTML
- libxslt for XSLT transformations (XML to HTML or text)
- Local caching of metadata
- Local database for offline access (work in progress)
- Generic view without schema dependencies
- Editor tailored to feature schema
- Validation for edited features (“business logic”)
- Printing via external tool (based on Docbook and FOP)

Fate Screenshot

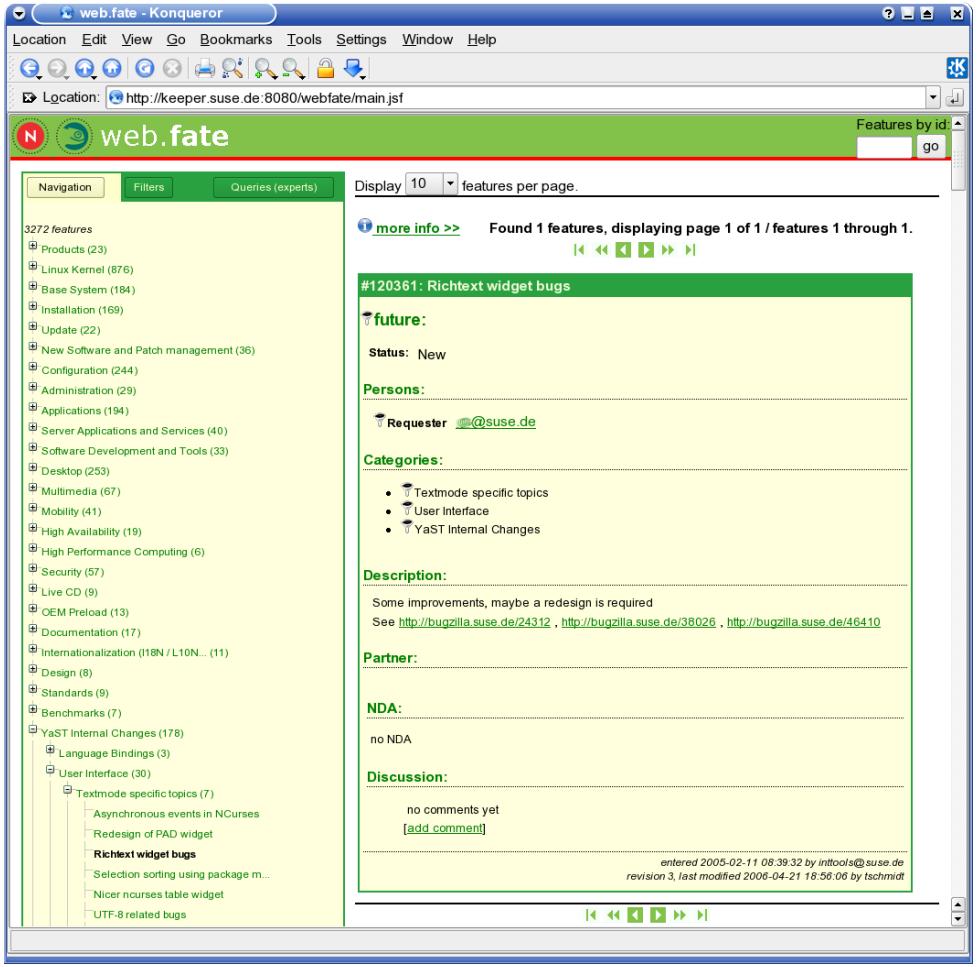


web.fate

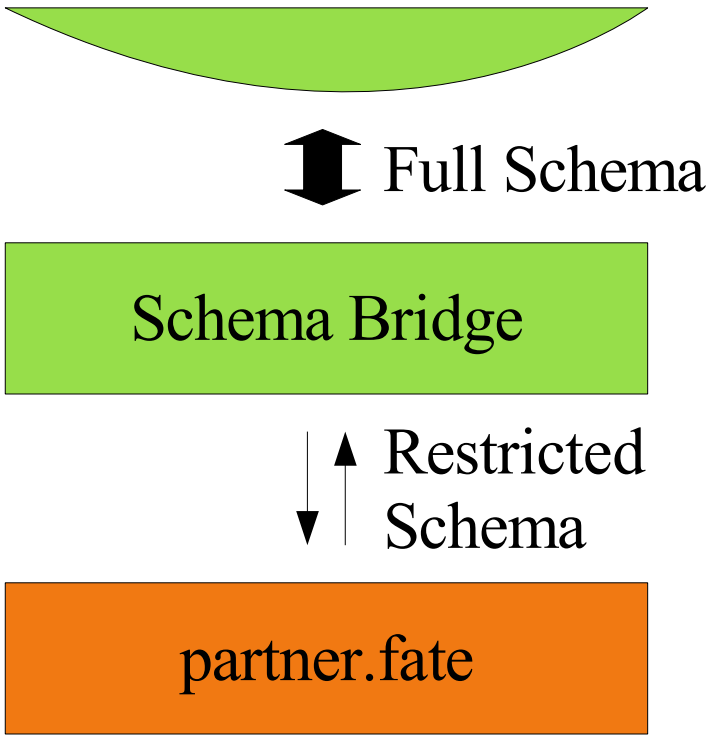


- Web Client
- Queries
- Browse
- Adding comments
- Java Server Faces
- Shares style sheets, etc. with Fate

web.fate Screenshot

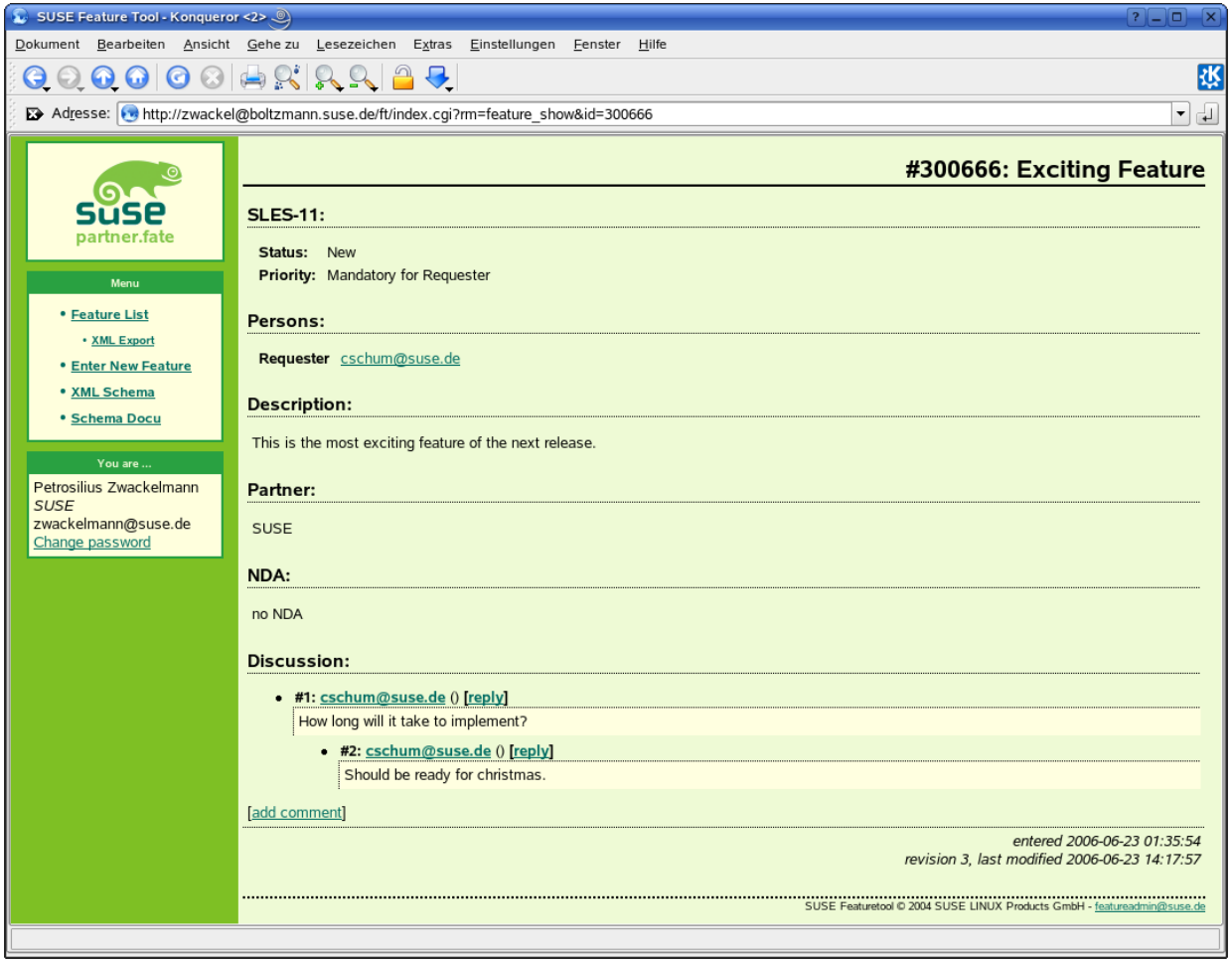


patner.fate



- Web client for limited access
- Restricted access to subset of data
- Limited view
- Limited edit rights
- Perl implementation

partner.fate Screenshot



Generic Clients

- Standard protocol allows to use standard clients

- Web browser to directly browse keeper
- HTTP aware network-transparent tools
  - kate for direct editing of raw XML data
- HTTP tools
  - curl
  - wget

- Writing clients is easy (guess why we already have three)

Demonstration

Conclusion

Conclusion

- Feature data schema
- Light-weight feature management process
- Client-server feature management system
- Generic HTTP and XML based server
- Several clients for different purposes

- Generalization to other schemas possible

- Open Source
  - Releases: Keeper: sxkeeper 1.0, Fate: fate 1.2
  - `developer.novell.com/wiki/index.php/Sxkeeper`
  - Mailing list: `sxkeeper-devel@forge.novell.com`

Novell®