Background

The Meta-Environment is an interactive development environment for

• defining domain-specific languages and the tools to support them, and

• performing analysis and transformation of existing software.

See www.meta-environment.org [http://www.meta-environment.org] for further details. We want to present our documentation in a more coherent fashion. The purpose of the current document is three-fold:

• to describe a general documentation policy for The Meta-Environment.

• to identify the documents that will be written or will be rewritten.

• to make a selection of tools and techniques that can be used to implement this policy.

Note

Updated October 30, 2006.

Table of Contents

Background ................................................................. 1
Current situation .......................................................... 2
The ideal situation .......................................................... 2
Requirements and design decisions .................................... 2
Audience ........................................................................ 3
Topics and presentation model .......................................... 4
Documents ..................................................................... 5
Learning about ............................................................. 5
How to ............................................................................ 5
Understanding ............................................................... 7
Courses ......................................................................... 8
Project management ....................................................... 8
Source code documentation ............................................ 8
How to get there? ............................................................ 9
Resources ..................................................................... 9
DocBook resources ........................................................ 9
Image resources ............................................................. 10
Plan ............................................................................ 10
Current situation

Currently, we use the following mechanisms to document (parts of) the system:

• Scientific papers that describe specific techniques or case studies.

• LaTex documents to describe either the whole system (e.g. user manual) or specific tools or libraries (e.g., aterm library, toolbus). HyperLatex is being used for generating web pages from LaTex files.

• POD files for manual pages.

• API documentation generated with javadoc.

• Web pages in our SEN1 twiki (with frequently asked questions)

The bad thing about the current situation is that it does not provide:

• Documentation on how to perform any specific task with the software.

• Technical documentation on how things work on an engineering level (we only explain things on a scientific level).

• Community support (the only help can be provided by the developers, who do not always have the time).

• Up-to-date documentation with respect to user-interface and syntax issues.

• An indexable overview of all documentation on the Meta-Environment.

• Online help.

The ideal situation

In an ideal situation we have a coherent plan for The Meta-Environment ’s documentation. This includes a common style, a model of our audience, a clear framework/structure for placing documents in, and a list of important documents.

Requirements and design decisions

We have two main desires that this documentation plan must fulfill:

• To facilitate the application of The Meta-Environment by more people to more applications. This implies better user documentation.

• To facilitate the maintenance and extension of the Meta-Environment by our open-source community. This implies better technical documentation.

We have identified the following requirements for our documents:

1. Ideally, we want to present all information about our system in a unified (read “predictable”) manner. Documentation can be organized in various manners, e.g. task-oriented, technology-oriented, user-oriented, developer-oriented, etc. Another critical decision is to use large, integrated, manuals or more loosely coupled information. The requirement is to be consistent in this organization.

2. We want the information to be targeted at specific audiences; better documentation is written for a (more or less) specific audience.
3. We want the documentation framework to be easily extensible (to minimize the effort of maintaining/updating the information, adding documents, etc.)

4. We want to publish the documents in different media (online, pdf, print).

5. We want an indexable overview.

6. We want low-level API documentation to be tightly integrated with the code (such that it can change or migrate with it).

7. We want documentation for specific use-cases, practical information on how to apply the technology to obtain a specific goal.

8. We want in-depth documentation on how the technology works and what the engineering trade-offs are.

So, we have taken the following design decisions:

- We will organize the documentation in short, loosely coupled, tightly integrated documents (R3,R4). Loosely coupled articles are good for web presentation, and can be combined with some additional glue in a paper form. Tight integration means that a single document describes one issue completely, albeit at a certain level of abstraction.

- We will use a common presentation style

**Audience**

Our audience consists of the following groups of people:

- **Users** are people that are not interested in the source code level of the software: they deal with the commandline and GUI interfaces, and nothing else.
  - Users of the ASF+SDF Meta-Environment.
  - ASF+SDF Meta-Environment newbies.
  - Users of SDF.
  - Users of the ATerm library and ApiGen.
  - Bachelor and master students.
  - Scientific colleagues working on related topics.
  - Scientific colleagues working on topics that could be applications of the Meta-Environment.

- **Developers** are interested in the source code level and want to know all technical details.
  - Developers of any of the components.
  - Advanced users (contributors of applications).

- **Decision makers** are interested in global scientific impact, innovation level and economic impact.
  - Project managers who may decide to use our technology in their projects.
  - Decision makers at funding agencies who can decide to fund research and development related to The Meta-Environment.
  - Decision makers in industry who can decide that using The Meta-Environment has business value for them.
We classify members of each group into maturity levels:

1. **Novice**: has expressed the intention of participating, does not know where to start, or where to end. Does not have an idea of scope. Has to deal with a new paradigm.

2. **Professional**: has adequate experience, has a certain overview of the system, knows the scope and limitations, can provide feedback to *Novices*.

3. **Specialist**: has in-depth understanding of one or more particular products, can provide feedback to *Professionals*.

4. **Owner**: is the representative of one or more particular products or applications, can provide feedback to and stimulate *Specialists*.

This classification helps us to think about the audience of a certain document. Each document should make clear at the beginning what the expected audience is.

### Topics and presentation model

There is a wide range of topics to discuss, including:

- High-level descriptions of purpose and architecture.
- User documentation (how-to's, demonstrations, frequently asked questions).
- Detailed technical explanations of the design, algorithms and engineering trade-offs.
- Interactive online demonstrations.
- Descriptions of successful applications and application areas.
- Open source project management issues (coding conventions, roadmaps).

Furthermore, we have some additional information that is probably more volatile:

- Download area with instructions.
- Description of the licenses we are using.
- Access to bugtracker.
- Access to SVN repository.
- Access to (archived) mailing lists.

We will use the following three-step presentation model:

- Learning about - overview of concepts and features of key components.
- How to - using key components and solve problems.
- Understanding - detailed knowledge about concepts and implementation.

Based on this model, we will use the following categories:

- Learning about - For novices and professionals.
- How to - For novices and professionals.
- Understanding - For specialists and owners.
- Courses - For Novice and Advanced Users.
• Project - For Developers.
• Source code/API documentation - For Developers.

Documents

Each category will be described and a number of documents to be written is identified. The idea is that these categories are easily extensible with new material, because the documents are relatively small. Still, this plan mentions a number of documents that need to be written to get things started (or rather to catch up).

Learning about ...

*Learning about* articles describe essential concepts, global application areas and give overviews of the features of components of the system. Typical subjects:

• Learning about the architecture of The Meta-Environment.
• Learning about software analysis
• Learning about software transformation
• Learning about domain-specific languages
• Learning about syntax analysis
• Learning about term rewriting
• Learning about SDF
• Learning about ASF
• Learning about ASF+SDF

*Learning about* articles:

• Target novices or professionals.
• Are amply illustrated.
• Gives pointers to the general literature as well as to Meta-Environment related documents.

How to ...

*How to* articles provide tutorials for various aspects of the software. These articles are written with a particular use-case in mind, and provide a domain specific view of the technology. The focus is on what you can do with it, and not on how it actually works. Typical subjects:

• How to download and install SDF (on linux, macos, windows)
• How to download and install the ASF+SDF Meta-Environment
• Howto downloading and install the ToolBus
• Howto start the ASF+SDF Meta-Environment
• Howto start the SDF Meta-Environment
• Howto use the basic SDF commandline
• Howto use the basic ASF commandline
• How to write a grammar in SDF
• How to write equations and functions in ASF
• How to fix a parse error
• How to fix an ambiguity
• How to analyse source code in ASF
• How to transform source code in ASF
• How to specify an algebra in ASF+SDF
• How to write a compiler in ASF+SDF
• How to write an interpreter in ASF+SDF
• How to use ToolBus to let tools written in different languages work together
• How to extend the Meta-Environment
• How to use the SDF commandline tools
• How to use the ASF commandline tools
• How to prototype a domain specific language in ASF+SDF
• How to use the generic debugger TIDE
• How to write parameterized ASF+SDF modules

The above articles may be cross-linked. Especially if there is a hierarchical dependency such as: 'How to analyse source code in ASF', which references 'How to write a grammar in SDF'.

How to articles:
• Start with a summary of tasks, including a work-flow diagram
• Target either novice or professional users
• Mention the necessary knowledge and experience of the user
• Written by either specialists or owners
• Reference Understanding documents where the information would become too technical (example: explain how to use SDF priorities and what for, but not how they work)
• Avoid the use of screen-shots, and other volatile details (provide a reference to a Demonstration)

How to apply articles:
• Short (1-3) pages
• Written by specialists or developers.
• Targeted at users
Documentation Plan for The Meta-Environment

- References to relevant websites and scientific publications
- Mentions domain (application area)
- Mentions software used
- Mentions future work
- Mentions all scientific and industrial partners
- Mentions all people involved
- The list itself is worth publishing first-class on the website (including the names of the authors)
- Each application is one docbook file
- If possible, each application has a corresponding Demonstration

**Understanding ...**

Articles in the category understanding give in-depth, detailed, information about concepts, technology or implementation. Typical subjects:

- Understanding ATerm data representation and manipulation
- Understanding SDF disambiguation and filtering
- Understanding ASF traversal functions
- Understanding ASF conditional term rewriting
- Understanding SDF grammar normalization
- Understanding source code locations
- Understanding SDF scannerless generalized LR parsing
- Understanding ToolBus coordination middleware
- Understanding the Relational Calculus
- Understanding TIDE, the language independent debugging interface
- Understanding the connection between ASF and SDF
- Understanding SDF parse trees
- Understanding ATerm API generation
- Understanding SDF API generation
- Understanding Source code tree composition
- Understanding Sisyphus modular continuous integration toolkit

Typically these documents have the following properties:

- Written for a more advanced audience (both developers and users)
- Provide references to scientific publications
- Arbitrary length (average 10-15 pages), but shorter is better!
Documentation Plan for The Meta-Environment

- Starts with explaining it's position relative to the rest of the software
- Provides plenty links to related work and related software.
- May also be a pointer to a scientific publication.

Courses

The courses section of the documentation contains slides and hand-outs for use in bachelor and master level courses on Meta-Environment related subjects. Since layout is very important for slides, we allow powerpoint, open-office and latex documents.

Project management ...

The project management category contains information for developers
- Documentation plan
- Communication plan
- Software process standard
- How to document source code
- How to use version control
- Coding standards
- Roadmaps (per main topic such as SDF, ASF, ToolBus, ATerms, Sisyphus)
- Testing plans
- Source code distribution and source code composition manual

Source code documentation

This is the documentation located with and inside the source code:
- The documentation is organized per package
- Every package will have a 'make doc' target, that will generate a web page with in index.html file in ./doc/html
- The documentation resides mainly inside the source code (such that it can migrate with it)
- Only one API, or one tool in one package, such that documentation is easier (work in progress)
- For C packages we use doxygen
- Fox Java packages we use javadoc
- For other languages we will extend doxygen with a DSL plugin (e.g. ToolBus, SDF)

For each package we need:
- A global description of purpose and main use cases (to be located in the MAIN HEADER FILE of a tool/library, using the \mainpage command of doxygen). This should reference to other documentation sources such as How to's, Understanding and Demonstrations
- A manpage-like description of command-line use of that package (to be located in the directory that contains the main() function of the tool, and written using docbook)
• Description of the APIs provided by the package (using the doxygen \groupdef command (although we strive to have only on API in one package)

How to get there?

It seems that the most widely used technology for documenting software projects is DocBook (see, www.docbook.org [http://www.docbook.org]). Docbook is an XML-based standard for the semantic mark-up of text and can be used to generate a wide variety of output formats including HTML, PDF, man pages, javahelp, and eclipsehelp. This means that we can use a single source for our webpages, interactive help and printed documentation.

The downside of using DocBook is that we have to learn yet another text processing technology. The number of predefined tags in DocBook is large (400) and the markup is more verbose than, for instance, LaTex. Another issue to observe is that DocBook is under active development. DocBook V4.5 is frozen and DocBook V5.0 is in an advanced stage of development. After experimenting with both versions, it seems better to start using V5.0 immediately, otherwise we start writing legacy documentation and need to convert later on.

We have tried a number of DocBook editors including:

• The xxe editor by XMLMind (www.xmlmind.com/xmleditor [http://xmlmind.com/xmleditor]). The standard edition of xxe is freely available.

• The Eclipse-based editor Vex (vex.sourceforge.net [http://vex.sourceforge.net]).

• The conglomerate editor (www.conglomerate.org [http://www.conglomerate.org]).

• The nXML extention of emacs (hacks.oreilly.com/pub/h/2044 [http://hack.oreilly.com/pub/h/2044]).

• The DocBook extension of openoffice (xml.openoffice.org/xmerge/docbook/index.html [http://xml.openoffice.org/xmerge/docbook/index.html]).

Based on this experience we recommend to use xxe for wysiwyg editing and emacs for editing of the XML source. Note that the DocBook V5.0 addon has to be installed explicitly after installation of xxe itself (Start xxe, go to the Options->Install Add-ons... menu and select either DocBook 5 + XInclude configuration or Customization of the DocBook 5 configuration allowing edit <xi:include> elements by hand).

Resources

DocBook resources

Useful links related to DocBook:


• The DocBook project (docbook.sourceforge.net [http://docbook.sourceforge.net])

Useful books related to DocBook are:


Image resources

Documentation becomes more lively and inspiring with good use of illustrations and pictures. Think of the following associations:

<table>
<thead>
<tr>
<th>analysis, parsing</th>
<th>measurement tools, knives, microscopes, lenses, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-defined syntax</td>
<td>masks, cameleons</td>
</tr>
<tr>
<td>transformation</td>
<td>various biological analogies (from seed to plant, metamorphosis of butterflies), electrical transformers</td>
</tr>
<tr>
<td>term rewriting</td>
<td>trees, forests, ...</td>
</tr>
<tr>
<td>software renovation</td>
<td>renovation of buildings, cities, ...</td>
</tr>
<tr>
<td>Software components</td>
<td>lego bricks, (toy) building blocks, electrical outlets, power adapters,</td>
</tr>
</tbody>
</table>

Here is a list of links to collections of potentially relevant images:

- Image databases of the University of Amsterdam: www.uba.uva.nl/dpc/beelddatabanken.cfm [http://www.uba.uva.nl/dpc/beelddatabanken.cfm]. In particular the botany and map collections are interesting.


- CalPhotos: calphotos.berkeley.edu [http://calphotos.berkeley.edu/].

- Masks: exchanges.state.gov/culprop/bolivia/sect1.htm [http://exchanges.state.gov/culprop/bolivia/sect1.htm].

Google image search is also highly recommended for finding relevant illustrations. Be consistent in your choice of illustrations (one theme per topic).

Plan

- Switch to DocBook for writing all documentation.
- Existing documentation will be converted to DocBook.
- Split up ATerm manual into pieces: transform API to doxygen, ToolBus howto for connection tools, and Understanding ATerms for the rest, maybe a small 'How to program ATerms'
- Existing javadoc documentation in Java will be rendered by javadoc Doxygen and included in the Twiki pages.
- For all non-java documentation, we will start to write Doxygen comment
- Use Doxygen to generate documentation for all non-java packages, and include in the Twiki pages. (DONE)
- Add a doc target to all our packages and include the documentation in the website.
- Man pages and API documentation will be included in each package.
- Write manpages in docbook format (in the directory that contains the main() of the tool).
- More global documents will be included in the meta-doc package.
- Publish meta-doc documentation in the Twiki pages (continuously)

• Start write How to and Understanding articles, after prioritizing them, reusing parts of the old documentation

• Replace Pico guided tour by interactive Demonstration

• Make packages smaller (one tool/library, one package)

• Invite people to write Applications documents: Ralph Oudejans (First result), Mark van den Brand (Risla, Elan4), Albert Hofkamp (Chi), Eelco Visser (StrategoXT), Pierre-Etienne Moreau (Tom), Peter Mosses (Action Notation), Jan-Friso Groote/Jaco van der Pol (muCRL), Anthony Cleve (Data reengineering).