



HAL
open science

D4.2 Conceptual Models of Integration of New Services

Carsten Thiel, Michelle Rodzis, Raisa Barthauer, Stefan Buddenbohm, Yoann Moranville

► **To cite this version:**

Carsten Thiel, Michelle Rodzis, Raisa Barthauer, Stefan Buddenbohm, Yoann Moranville. D4.2 Conceptual Models of Integration of New Services. [Technical Report] Göttingen State and University Library. 2018. hal-01825919

HAL Id: hal-01825919

<https://hal.science/hal-01825919v1>

Submitted on 28 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



D4.2 Conceptual Models of Integration of New Services

DESIR

DARIAH ERIC Sustainability Refined

INFRADEV-03-2016-2017 - Individual support to ESFRI and other world-class research infrastructures

Grant Agreement no.: 731081

Date: 28-06-2018

Version: 1.0



DESIR has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731081.

Grant Agreement no.:	731081
Programme:	Horizon 2020
Project acronym:	DESIR
Project full title:	DARIAH-ERIC Sustainability Refined
Partners:	DIGITAL RESEARCH INFRASTRUCTURE FOR THE ARTS AND HUMANITIES GEORG-AUGUST-UNIVERSITAET GOETTINGEN STIFTUNG OEFFENTLICHEN RECHTS UNIVERSITEIT GENT UNIWERSYTET WARSZAWSKI FACULDADE DE CIENCIAS SOCIAIS E HUMANAS DA UNIVERSIDADE NOVA DE LISBOA CENTAR ZA DIGITALNE HUMANISTICKE NAUKE GOTTFRIED WILHELM LEIBNIZ UNIVERSITAET HANNOVER INSTITUT NATIONAL DE RECHERCHE ENINFORMATIQUE ET AUTOMATIQUE KING'S COLLEGE LONDON UNIVERSITY OF GLASGOW KNIHOVNA AV CR V. V. I. HELSINGIN YLIOPISTO SIB INSTITUT SUISSE DE BIOINFORMATIQUE UNIVERSIDAD NACIONAL DE EDUCACION A DISTANCIA UNIVERSITY OF HAIFA UNIVERSITY OF SHEFFIELD (joining) UNIVERSITY OF NEUCHATEL (joining)
Topic:	INFRADEV-03-2016-2017
Project Start Date:	01-01-2017
Project Duration:	36 months
Title of the document:	Conceptual Models of Integration of New Services
Work Package title:	Technology

DESIR

INFRADEV-03-2016-2017 - Individual support to ESFRI and other world-class research infrastructures, Grant Agreement no. 731081.



Estimated delivery date:	30-06-2018
Lead Beneficiary:	UGOE-SUB
Author(s):	Raisa Barthauer [barthauer@sub.uni-goettingen.de] Stefan Buddenbohm [buddenbohm@sub.uni-goettingen.de] Carsten Thiel [thiel@sub.uni-goettingen.de]
Quality Assessor(s):	Robert Jäschke [robert.jaeschke@hu-berlin.de] Yoann Moranville [yoann.moranville@dariah.eu]
Keywords:	DARIAH, research infrastructure, sustainability, technology, technical reference, software quality

Revision History

Version	Date	Author	Beneficiary	Description
0.1	01.06.2018	Raisa Barthauer	UGOE	Outline and References
0.2	08.06.2018	Carsten Thiel	UGOE	First Draft
0.3	15.06.2018	Stefan Buddenbohm Carsten Thiel	UGOE	Revision
1.0	26.06.2018	Carsten Thiel	UGOE	Final Version, including QA

Executive Summary

A fundamental basis of a successfully operating digital infrastructure such as DARIAH is formed by the services it provides to its users. In the particular case of the distributed setup DARIAH is using, the integration of new services requires support and guidelines that can be agreed to by all current and future service providers. Such generic guidelines can support individual research as well as new research projects just starting out, and – ideally – later enable the infrastructure to sustain their products.

Nature of the deliverable	
R	Document, report
✓ DEM	Demonstrator, pilot, prototype
DEC	Websites, patent fillings, videos, etc.
OTHER	
Dissemination level	
P	Public
✓ CO	Confidential only for members of the consortium (including the Commission Services)
EU-RES	Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)
EU-CON	Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)
EU-SEC	Classified Information: SECRET UE (Commission Decision 2005/444/EC)

Disclaimer

DESIR has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731081. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Introduction

Through DESIR (DARIAH ERIC Sustainability Refined), DARIAH strives to define a roadmap for the sustainable operation of its services. This consists of the administrative sustainability through a business plan as well as guidelines and best practices for the actual development and operation of the services. The latter is addressed by work package 4 “Technology”, aiming to technologically enhance the DARIAH research infrastructure and its services. The implementation of new services into the DARIAH infrastructure has been already investigated within the Humanities at Scale¹ project (see Matoni, Schmunk, Thiel 2017). One of the specific challenges within DARIAH is its distributed nature:

[...] the DARIAH infrastructure does not have a central coordinating authority that decides on its portfolio and strategic plans to extend and improve its overall possibilities. Rather, the DARIAH member countries contribute the services their partner institutions operate to form the DARIAH infrastructure. Therefore, the decision to integrate a service is always up to random volunteers addressing national requirements. (Matoni, Schmunk, Thiel 2017, Sec. 4)

Building on this previous work, DESIR has organised a workshop with service providers and representatives of DARIAH and related infrastructures in Berlin in October 2017 (cf. Thiel 2017) to discuss these matters. Amongst the participating infrastructures CESSDA², CLARIAH³, CLARIN⁴ and DARIAH, the problems are similar and a number of commonalities were identified. These range from the requirements of service providers to use supported and supportable technologies to the user’s requirement for continued availability of the services they need. These contrast with the individual research engineer’s goal of developing innovative technology and building a demonstrator that can be used for new academic publications. At the same time, as technology and research questions develop and progress, new requirements can emerge to continue development of individual solutions. Thus, one of the central tasks for DESIR identified at the workshop was the formation of developer guidelines as basis for reusability:

[These needs] can possibly be addressed through guidelines, recommendations, and criteria catalogues at varying levels of detail. Elementary guidelines for software developers just starting out, which typically only involve modest scripting and ad-hoc solutions implemented for highly specialised problems, can support the process from the start. (Thiel 2017)

¹ Humanities at Scale, Horizon 2020, Project ID 675570

² <https://www.cessda.eu/>

³ <https://www.clariah.nl/en/>

⁴ <https://www.clarin.eu/>

The Technical Reference

The DARIAH Technical Reference (TR), is available online (Thiel, Rodzis, Moranville 2018) as well as an annex to this document. It is being developed within DESIR and under its coordination and includes external contributions. The TR builds on a number of previous guidelines and best practices developed by several of the workshop attendants, in particular the CLARIAH Software Quality Guidelines (van Gompel et al 2016), the Netherlands eScience Center Guide (NLeSC 2018) and the CESSDA Software Maturity Levels (Shepherdson et al 2016).

The purpose of the reference is to collect best practices and software development guidelines as well as quality checklists. It is a collection of best practices and guidelines for developers and maintainers of infrastructure components, as well as quality recommendations. They can be used to either gauge the quality of ongoing developments or as a starting point for new research projects, in particular when the components are planned to be attached to the infrastructures.

The TR is a reference in that it lists general requirements and considerations, but it does not always specify choices. In particular, it does not define licences, technology stacks or hosting services. These are part of the implementation of the TR for a research infrastructure or a data centre. This approach was chosen, because it allows an increased compatibility with existing (internal) requirements on specific choices and it reduces the effect of the ‘not invented here’ syndrome⁵.

Further Work

Over the course of the remaining project time, the applicability of the TR to software development will be tested, in particular at the DESIR code sprint⁶. Applying the guidelines and best practices to the software developed by DARIAH and its partners in projects will ultimately allow to create new DARIAH services. Ideally, this will enable DESIR to create new services addressing the gaps identified previously, see Buddenbohm, Barthauer 2017, who provided an overview of the existing DARIAH infrastructure and services.

At the same time, quality criteria play a vital role in DARIAH’s assessment of its partners’ contributions as well as in the upcoming SSH Open Marketplace⁷. The Marketplace will be a registry of services and resources and will be based on certain quality standards for the

⁵ https://en.wikipedia.org/wiki/Not_invented_here

⁶ To take place in Berlin, 31.07.–02.08.2018, <https://desircodesprint.sciencesconf.org/>

⁷ The Marketplace has been identified as high priority in the DARIAH Strategic Action Plan. DESIR WP5 will release a mockup of the Marketplace, work is planned for future DARIAH activities.

listing of services and resources. But it is also clear that the quality standards for the registry will likely be oriented to a large part at end users of the DARIAH infrastructure and services and whereas the TR addresses the development phase and operational aspects of services.

Beyond the immediate work within DESIR, the collaboration among CESSDA, CLARIN and DARIAH continues on the technical level as well as other common areas that have been identified by the governing bodies.

References

Stefan Buddenbohm, Raisa Barthauer. 2017. D4.1 - Gap Analysis of DARIAH Research Infrastructure, <https://hal.archives-ouvertes.fr/hal-01663594> [Last accessed June 26, 2018]

Maarten van Gompel, Jauco Noordzij, Reinier de Valk, Andrea Scharnhorst. 2016. CLARIAH Software Quality Guidelines, version 1.1, September 30, 2016, available at <https://github.com/CLARIAH/software-quality-guidelines> [Last accessed June 26, 2018]

NLeSC, 2108, Netherlands eScience Center Guide, Available at <https://guide.esciencecenter.nl/> [Last accessed June 26, 2018]

John Shepherdson, Ørnulf Risnes, Mike Priddy, Matti Heinonen, Johan Finn, Wolfgang Zenk-Möltgen. 2017. CESSDA Software Maturity Levels (v1.0) available at <https://drive.google.com/file/d/0Bwk0RK5TD06iTWxrTXFGSG5tMzg/view> [Last accessed June 26, 2018]

Markus Matoni, Stefan Schmunk, Carsten Thiel. 2017. Basic DARIAH Services and Demonstrators <https://hal.archives-ouvertes.fr/hal-01574981> [Last accessed June 26, 2018]

Carsten Thiel. 2017. Workshop „Software Sustainability: Quality and Re-usability“, available at <https://dhd-blog.org/?p=8685> [Last accessed June 26, 2018]

Carsten Thiel, Michelle Rodzis, Yoann Moranville. 2018. DARIAH Technical Reference, available at <https://dariah-eric.github.io/technical-reference/> [Last accessed June 26, 2018]

Technical Reference

Release 0.1

Carsten Thiel, Michelle Rodzis, Yoann Moranville

Jun 28, 2018

CONTENTS:

1	Developer Guidelines	3
1.1	Basics	3
1.2	The README	3
1.3	Documentation	4
1.4	Tooling	5
1.5	Interoperability	6
1.6	Changelog	6
2	Operational Guidelines	7
2.1	Basic Operational Guidelines	7
2.2	Infrastructure Documentation	7
2.3	Documentation for (virtual) servers	7
2.4	Security	8
2.5	Incidents and Postmortems	8
3	Policy Recommendations	11
3.1	Code Hosting Policies	11
3.2	Release policy	12
4	Software Quality Checklist	13
4.1	General	13
4.2	Documentation	13
4.3	Development	14
4.4	Interoperability	14
4.5	Administration	14
	Glossary	15
	Bibliography	17

This is the technical reference for the DARIAH Infrastructure. It is a collection of basic guidelines and references for development and maintenance of infrastructure services within DARIAH and beyond. This work relies heavily on work by the community and in particular [CESSDA ERIC](#), [CLARIAH-NL](#), [DARIAH-DE](#) as well as the [Netherlands eScience Center](#) and the [UK Software Sustainability Institute](#).

This work explicitly relies on the [Netherlands eScience Center Guide](#) and the [CLARIAH Software Quality Guidelines](#).

To make use of these guides, implementing an institutional manual based on this is recommended. In many places, possible or required choices are pointed out and should be made for a specific manual.

It is particularly recommended to adopt the [Recommendations to encourage best practices in research software](#) whenever starting a new project. Also, consider using the [Software Sustainability Institute's Software Management Plan](#).

This work is licensed under [Creative Commons Attribution 4.0 International](#).

DEVELOPER GUIDELINES

1.1 Basics

All development should be made available publicly under [open source](#) licences.

1. Use version control right from the beginning of a new project.
 - If in doubt, use [Git](#).
 - Implement a *Code Hosting Policy*.
 - Use meaningful commit [messages](#), cf. [\[ProGit\] Sec. 5.2](#):
 - Capitalised summary with a maximum of 50 characters followed by a blank line.
 - Detailed but concise explanations in paragraphs or bullet points at 72 characters line length.
2. Use an appropriate [OSI](#) approved license.
 - Decide on an appropriate license before you first commit.
 - Ensure the license is compatible with all dependencies.
 - If in doubt, choose [APACHE-2.0](#) or [EURL-1.2](#).
 - Add the text in a `LICENSE.txt`.
3. Maintain a *README*.
4. *Document* your software properly.
5. Use existing *tooling* to support development workflows.
6. Ensure maximal *interoperability*.
7. Ensure your software is usable and accessible.
8. Implement a *release policy* and keep a *changelog*.
9. Add a code of conduct in a `CODE_OF_CONDUCT.md`, like we do.
10. Specify contribution policies in a `CONTRIBUTING.md`, like we do. A legitimate policy can be that external contributions are not accepted and merged.

1.2 The README

The first thing users and other developers look at when first making contact with a repository is the `README.md` in the repository's root directory. Its purpose is to give a concise but comprehensive introduction to the project. It should

provide links to further (more detailed) documentation, websites or other background information. Depending on the relevant ecosystem, specific guides or templates for READMEs exist.

The following basic information must be provided:

1. The project name
2. A short but meaningful descriptive summary of the repository
3. The maintenance status

The following questions must be answered:

1. What does the software do?
2. Who will use the software?
3. What are alternatives to the software, and how do they differ?
4. How can someone get started?
 - Requirements
 - Binary download location
 - Build instructions with dependencies
 - Installation instructions
 - Quick start examples
 - Link to full user documentation
5. How can others join the development?
 - Coding styles used
 - API design
 - Toolchain and frameworks used
 - Community communication platforms
 - Link to developer documentation
 - Test suite details
 - Contribution guides
6. Release details and versioning
7. Who has contributed to the software?
8. Which license is used? In most cases a link to the *LICENSE.txt* is sufficient.
9. Who has to be acknowledged? I.e. who has played a significant role for the creating of the software? This can e.g. include funding, research communities, or co-workers that are not part of the project but have given advice and/or input to the development process.

1.3 Documentation

Documentation is fundamental to ensure usability and usefulness of the software. It must be stored along the code, ideally in the repository's `docs` folder. Basic documentation should also be included in the *README*.

Documentation is relevant in many forms, each of which should be addressed for different audiences with varying degree of experience and knowledge.

1. **User documentation:** Include a documentation for end users, including e.g.
 - Examples
 - Tutorials
 - How-Tos
 - FAQs
 - Screen-casts
 - API documentation
2. **Developer documentation:** Provide instructions for developers.
 - How to set up the environment.
 - Dependencies, including
 - Supported operating systems
 - Required libraries
 - External dependencies
 - Requirements, e.g. hardware, architecture, CPU, RAM, disk space and network bandwidth.
 - How to build the code.
 - How to package the code.

Additionally, inline code documentation should be used as appropriate.

- Always adhere to the language's standard or well established one such as the [Google Style Guides](#).
 - Document the *why* and not the *what*, cf. [\[CleanCode\]](#).
3. **Administration documentation:** Provide instructions for installation, configuration and maintenance, in particular when as a daemon (e.g. a micro service).
 - Configuration instructions
 - Start-up script (e.g. `init` or `systemd`)
 - Monitoring setup, ideally through a monitoring endpoint

1.4 Tooling

In order to support an efficient development workflow and ensure a high degree of software quality the use of appropriate tooling is strongly recommended.

This should include:

1. Using an appropriate code editor or IDE.

While the individual person should be free to choose the solution that best fits his or her need, the editor/IDE should provide standard features such as syntax highlighting and code completion for all relevant languages.

Use `EditorConfig` to ensure consistency of code submitted by all developers. In case others can contribute to the software as well, it is recommended to add the respective `.editorconfig` to the software's repository.

2. Code linting within the editor, as pre-commit hooks and part of further automation should be used.
3. Unit testing to improve code quality and simplify future development.

4. Static code analysis to reduce common errors and improve overall quality by adhering to standards and best practices.
5. A Continuous Integration solution that runs on every code commit to verify the test suite, lint code, perform static analysis and more.

Popular choices for CI are:

- [Travis CI](#)
- [GitLab CI](#)
- [Jenkins](#)

1.5 Interoperability

1. Internationalisation principles must be applied to ensure future localisation.
2. Localisation of the software must be available in English and should be available in other relevant languages.
3. Provide an API.
4. Use federated authentication.
5. Provide full configuration examples.
6. Provide endpoints for monitoring (service status) and statistics (user request and hardware utilisation).

1.6 Changelog

1. Adopt a *release policy*.
2. Maintain a manually curated [Changelog](#).
 - Use a `CHANGELOG.md`, like `ours`.
 - Structure in a descending order by version with release date.
 - Add all changes applied in that version.
 - Have an *unreleased* section at the top for the next release.

OPERATIONAL GUIDELINES

For a generic collection of IT Service Management documents, see e.g. *[FitSM]*.

2.1 Basic Operational Guidelines

1. Maintain a proper up-to-date *documentation*.
2. Establish processes for recurring actions.
3. Automate as much as possible.
4. Implement and verify backups.
5. Monitor the infrastructure.
6. Consider software configuration management.

2.2 Infrastructure Documentation

1. Keep an up to date inventory of all infrastructure components used, including
 - Actual physical hardware (servers, switches, racks, tapes, appliances etc.)
 - *Virtual machines*
 - Cloud and container infrastructure
 - Storage solutions and instances
 - Backup systems
2. Keep an inventory of all services that are offered and their dependencies with each other and the underlying infrastructure.
3. Implement *security*.
4. Keep a record of *incidents*.

2.3 Documentation for (virtual) servers

Documentation for virtual servers should include the following information:

- Where is it?

- Location, e.g. rack number or visualisation infrastructure
- IP address(es)
- DNS name(s)
- What is it?
- Operating system
- Human readable purpose
- Who knows about it?
- Responsible system administrator
- Secondary system administrators
- Project manager
- Which services does it run?

For each service, list the following:

- Service name
- Ports
- Status verification
- Log file
- Configuration file(s)
- Data directories
- Dependencies
- How can it be fixed?

List possible procedures for emergencies, such as

- Whether to try rebooting or remounting something
- How to restore from backup

2.4 Security

Implement a good AAI and adhere to standards such as *[Sirtfi]* and *[Snctfi]*

2.5 Incidents and Postmortems

Record all outages including

- Which service was *disrupted*?
- What else was *affected*?
- Who was *in charge* of the recovery?
- *When* was the incident *discovered*?
- *How* and *by whom*?
- When has the incident *begun*?

- When was the incident *mitigated*?
- Who was *informed* and how?
- Has this ever happened *before*?
- Has *sensitive* data, such as user data or secrets, been compromised?

Particular importance should be applied to record all steps taken to mitigate the incident. These should include the person, time and specifics of any action taken.

Security breaches and vulnerability exploits may need to be reported to authorities, in particular if sensitive and/or (legally) protected data was (potentially) affected. Users must be informed appropriately, responsibly and quickly.

Finally, decide upon and implement measures to prevent repetitions.

POLICY RECOMMENDATIONS

3.1 Code Hosting Policies

3.1.1 General Rules

Make sure to publish your code in a version control repository.

- There are a number of well-known commercial solutions, such as
 - [GitHub](#)
 - [GitLab](#)
 - [Bitbucket](#)

They all offer some free options and using them has a number of advantages, e.g.

- Good and established usability
- High visibility of your code
- Low barrier for find-ability and re-use
- Good integration with other services and solutions

When using commercial and in particular external services, you must have a backup and data extraction strategy in place, which ensures that you can always move to another solution.

- There are a number of possibilities to host your own solution
 - The commercial solutions above.
 - [GitLab Community Edition](#)
 - [Gogs](#)
 - [Gitea](#)
 - [gitolite](#)

3.1.2 Specific Solutions

On Site

- Be sure to implement all relevant operational procedures.

GitHub

When using GitHub, implementing a suitable policy is recommended. This should include the following considerations.

Organisations

GitHub organisations can be used to group all repositories of an institution or a research project. It should be clearly stated who is responsible for an organisation, ideally stated on the organisation's GitHub Page.

Always ensure rights are managed by a sufficient number of people:

- At least two people should be owner of the organisation. It may be appropriate to have two owners from each institution involved in the project.
- Create an institutional account. This account becomes owner of all organisations the institution is involved in and makes sure access is granted to all appropriate individual employees.

Organisations should have a policy on who can be a member as well as how repository maintainers and maintenance status are defined and communicated.

Backup

As GitHub is owned by a commercial company (the Microsoft Corporation), the service provided through GitHub is subject to change through corporate development. It is highly recommended to set up an automated backup system, in order to ensure that a copy of all code and metadata (including issues, wikis etc.) exists.

Features

Use the features provided by GitHub, such as issue labels, issue and pull-request templates etc.

3.2 Release policy

- Use [semantic versioning](#).
- Provide releases as downloads.
- If applicable, provide binaries for all supported platforms.
- Have a roadmap.
- Consider providing releases via research repositories with citable references.

SOFTWARE QUALITY CHECKLIST

4.1 General

- Does the software have a descriptive name?
- Is there a short high-level description of the software?
- Is the purpose of the software clear?
- Is the targeted audience of the software clear?
- Does it (and its dependencies) use OSI approved licenses?
- Is the software under version control?
- Is there a website for the software?
- Does the software have a release mechanism?
- Is the software available in packaged format or only sources?
- Are maintainer and development status clear, including contact information?
- Are the requirements listed and up to date?
- Is the interface responsive and accessible?
- Is copyright and authorship clear?
- Is there a contribution guide?

4.2 Documentation

- Is there an accessible getting started guide?
- Is there an accessible user guide?
- Is there a full user documentation?
- Does the user interface link to held references?
- Are there examples, FAQs and tutorials?
- Are known issues documented?

4.3 Development

- Is the development setup documented?
- Is the build mechanism documented?
- Does the build mechanism use a common single-command system (i.e. Maven)?
- Is the software API documented?
- Are all appropriate config options externalised and documented?
- Does the code allow internationalisation (i18n)?
- Is the software localised (l10n)? English is mandatory.
- Is there a test suite?
- Is test coverage above 80%?

4.4 Interoperability

- Are file formats standard compliant and documented?
- Is the API standard compliant?
- Does it provide a monitoring endpoint?
- Does it adhere to an interface style guide?
- Does it use existing authentication systems (OAuth2/eduGain)?

4.5 Administration

- Are software requirements such as operating system, required libraries and dependencies specified including versions?
- Are hardware requirements for CPU, RAM, HDD, Network specified?
- Are there deployment instructions?
- Is there a comprehensive and fully documented example configuration?
- Is a start-up script provided?
- Are there troubleshooting guides?

GLOSSARY

These are some of the most important terms used throughout. See also [GitHub Glossary](#).

- *Collaborator* – Someone with write access to the repository.
- *Contributor* – Someone who submits code to the repository, either directly or via pull-/merge-request.
- *Developer* – Anyone who writes code.
- *Maintainer* – The person responsible for feature development and deciding about contributions.

BIBLIOGRAPHY

- [CleanCode] Robert C. Marton: **Clean Code: A Handbook of Agile Software Craftsmanship**, Prentice Hall PTR, 2008
- [FitSM] ITEMO: **FitSM** – A free standard for lightweight ITSM, <http://fitsm.itemo.org/>, 2016
- [ProGit] Scott Chacon, Ben Straub: **Pro Git**, <https://git-scm.com/book/en/v2>, Version 2.1.64, 2018-06-01
- [Sirtfi] AARC: **Security Incident Response Trust Framework for Federated Identity**, <https://aarc-project.eu/policies/sirtfi/>, 2015
- [Snctfi] AARC: **Scalable Negotiator for a Community Trust Framework in Federated Infrastructures** <https://aarc-project.eu/policies/snctfi/>, 2017