LRN2CRE8

LEARNING TO CREATE
PROJECT NUMBER: 610859

SMALL OR MEDIUM-SCALE FOCUSED RESEARCH PROJECT
ICT - FUTURE AND EMERGING TECHNOLOGIES (FET)

Deliverable 7.1: Technical Infrastructure

QMUL & Jožef Stefan Institute (JSI)[1] Version: 1.0, FINAL

Executive summary This deliverable proposes the software infrastructure and the data representation and exchange format for the Lrn2Cre8 project, enabling partners to build creative learning systems that can cooperate across the Internet, so that Lrn2Cre8 partners can collaborate efficiently among themselves, and with partners in the ConCreTe project where appropriate. To this end, the underlying technology is based on the concept of Web service architecture, allowing software developed at partner institutions to interact directly via well-defined APIs exposed in a standard way. This report defines the proposed extensible information exchange specifications, so that unforeseen issues may be addressed later in the project. The proposed representation is based on the RDF framework [6], enabling models and data to be shared directly among the partners. This report summarizes the efforts and achievements so far, aimed at ensuring systematic data management, sharing and processing.

Key elements of Lrn2Cre8's data management and processing are reflected in the following components: (I) the baseline: SOA architecture and RESTful[2] interfaces, and (II) the proposed RDF Lrn2Cre8 data representation format. This approach will allow the possibility for Lrn2Cre8 to deploy the ClowdFlows browser-based workflow construction and execution platform with its cloud-based memory facilities, if and when appropriate.

It is expected that the model and data representations will be further refined, changed and adapted to new findings, agreements and artifacts arising from the progress of the project. If needed, this will result in an update of this deliverable.

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	-
RE	Restricted to a group specified by the Consortium (including the Commission Services)	-
CO	Confidential, only for members of the Consortium (including the Commission Services)	-

[1] QMUL and JSI have collaborated on this document to avoid discrepancy and to promote collaboration between Lrn2Cre8 and ConCreTe, another FET STREP led from QMUL, in which JSI is a partner.

[2] RESTful services are services implemented using REST.

Contents

1 Introduction

The Lrn2Cre8 project aims to study the relationship between learning and creativity in the context of music. To facilitate interaction among various software components that will be developed in this pursuit, a common framework for deployment and interaction is used. A common data representation is also beneficial in machine learning projects, because it allows sharing of data within and outside the project; the aim here is also to enable leverage of the Clowdflows system, developed at the Jožef Stefan Institute, should it become appropriate.

In this document, the technical infrastructure of Lrn2Cre8 is defined and initial specifications for project's software components and data representations are provided. Some proposed specifications that are not mandatory for proper operation of the software system of Lrn2Cre8 are presented as recommendations, to emphasize that they may be violated, if there is a good reason or a strong preference for an alternative. It is expected that some of the recommendations will become specifications and that the specifications will have to be updated as the project progresses.

The project will use Web services as shareable workflow components. Data representations used will be compliant with the CHARM system [16, 13, 15], and an RDF description corresponding with the CHARM datatype is planned, enabling the work to be exported to the Semantic Web.

The document is structured as follows: in Section 2, the general architecture of the Lrn2Cre8 software platform is introduced and the platform of choice is presented. Section 3 provides proposed specifications of data representation in Lrn2Cre8, with particular focus on RDF as the data representation to which we aim to in Lrn2Cre8. Finally, in Section 4 a summary of specifications is provided in order to serve as a quick reference. Additionally, Appendix A lists the parameters and characteristics of software components that have to accompany any newly developed software component of Lrn2Cre8 in order to allow for a proper integration into the Lrn2Cre8 software platform and Appendix B provides an example of the specification of parameters and characteristics of a software component.

2 Service architecture of Lrn2Cre8

This section presents the service-oriented architecture of the Lrn2Cre8 project's technical infrastructure. Section 2.1 introduces service-oriented architecture in general terms. Section 2.2 discusses platforms for workflow construction and executions and their relations to service-oriented architecture. Section 2.3 outlines the main requirements for the Lrn2Cre8 architecture.

2.1 Service oriented architecture

Service-oriented architecture (SOA) [3] is a paradigm that specifies the design and implementation of software through the use of services, which are connected to each other and interact together. SOA is a distributed architecture that allows the user to build an application by means of composing individual components that exist across separate physical or logical domains. A service is a unit of clear and distinct functionality, independently deployed, which communicates solely through contractually specified interfaces, and only has explicit dependencies.

Services encapsulate processing logic within some context which can be a specific process step, sub-process, entire process or even logic provided by other services. As the services are required to interact, they rely on service descriptions which establish names and locations of services, as well as their data exchange requirements. Service descriptions ensure loose coupling of services, but a suitable communication framework is also required. Services commonly exchange autonomous units of communication through messaging.

When implementing a service-oriented architecture different technologies can be used, and the implementation usually relies on a set of related standards that form a compatible technology stack. The most commonly used technologies are:

- SOAP (Simple Object Access Protocol)[3], a protocol for exchanging structured information, which relies on XML for the message format and the HTTP protocol for message transmission,
- REST (REpresentational State Transfer) [4], a software architecture which is not as strongly typed as SOAP and whose language is based on the use of nouns and verbs with the emphasis on readability,
- WSDL (Web Services Description Language)[4], a machine-processable format in which the public interface for Web services is described.

In practice, one usually chooses between SOAP and REST when implementing the service-oriented architecture. When SOAP is used, the services are described with WSDL. Since version 2.0, WSDL can also be used to describe REST services.

Web-services are agnostic of the platform and programming language of the implementation of the service. They thus provide the means for communication between the software developed using heterogeneous technologies. The computation is distributed among servers hosting the services. Computationally intensive services can be hosted on dedicated high performance hardware, whereas simpler, less computationally expensive services can be bundled together on the same server. The complexity of the underlying implementation is hidden from the user. This improves scalability of the solution as it provides easier migration to more powerful hardware as the utilization of the service grows. In addition, the latest versions of underlying software libraries are provided automatically to all clients given that the services are updated regularly.

2.2 Platforms for scientific workflows

Construction of analytic workflows has been the topic of substantial research and development. For example, modern knowledge discovery systems offer means for workflow construction and execution. This is crucial for conducting complex scientific experiments, which need to be repeatable and easily verified by referees or peers. Workflows exploit the composability of services, assembling complex applications by reusing existing elementary services.

A well-known system is the Triana [8] workflow environment for P2P and Grid containing a system for integrating various types of middleware toolkit. Another prominent system is the Taverna [5] environment for workflow development and execution. Both Triana and Taverna support integration of SOAP and RESTful Web services within the workflows.

Within the knowledge discovery community, there are several well-known frameworks for workflow construction, management, and execution including KNIME [1], RapidMiner [10], and Orange [2]. In their newest incarnations, they all have implemented support for web-services. However, they are all implemented as standalone applications and have specific hardware and software requirements.

2.3 Requirements for the Lrn2Cre8 architecture

An important technical task of the Lrn2Cre8 project is establishing a protocol which will facilitate collaboration and sharing of developed resources and software. A key requirement for this protocol is the support for distributed development of software components. These components may require certain hardware and depend on particular software libraries. Software components will be developed by different project partners, where each partner can focus on components specific to the partner's expertise. The developed software components should be as easily as possible reusable by other project partners, without dealing with complexities of installation and configuration. For these reasons, where possible and appropriate, the components developed in Lrn2Cre8 will be provided as Web services.

[3] http://www.w3.org/TR/soap/
[4] http://www.w3.org/TR/wsdl

Another key aspect of collaboration is sharing of workflows and results. In an ideal world, the most suitable solution is a Web-based platform which would host all the workflows developed during the project. This will enable researchers to share workflows by simply providing their URLs. Such workflows can be executed by different users to recreate the same results. However, in a fundamental research project like Lrn2Cre8, this is not always possible. Therefore, development of workflows is not a key outcome of the project, but, instead, we propose to take advantage of our collaboration in the related ConCreTe project to gain access to the ClowdFlows [7] platform, developed by our ConCreTe colleagues in the Jožef Stefan Institute, Ljubljana. ClowdFlows runs as a Web application and places no software requirements on the user, other than a standards-compliant Web browser. It supports development of interactive scientific workflows which can be composed, inspected and executed, using the platform's Web interface.

3 Data exchange and representation

This section provides a more formal outline of data and model representations in the service oriented infrastructure of the Lrn2Cre8 project. It consists of three sub-sections: Section 3.1 introduces current data and model representation requirements of the ClowdFlows platform. The proposed representations are described in detail by Wiggins et al. [16, 13, 15], and summarised in Section 3.2.

3.1 Platform requirements

ClowdFlows, the underlying platform of ConCreTe's software infrastructure and also potentially for Lrn2Cre8, supports two types of components:

- native widgets and
- Web services.

It is expected that computational creativity software components that will arise from Lrn2Cre8 will mostly be implemented as Web services, particularly in the first implementation stage. Ideally, among them, a very generally useful and applicable set of software components would be identified and re-implemented as native widgets. These would form a new *Computational Creativity* set of widgets that would become a new module available as a part of the standard ClowdFlows distribution.

3.1.1 Native widgets

Native widgets reside on the server where the ClowdFlows platform is installed. They are written in Python and communicate their inputs and outputs in any data type supported in Python, such as integer, float, string, list, dictionary, object, etc. To be even more specific, the widgets use serialized data files for exchanging input/output. The Python *pickle*[5] module is used for serialization and the widgets can exchange data of any Python type that can be serialized with *pickle*. This includes most of the usual data types. Very few data types cannot be pickled and consequently exchanged among widgets, such are for example file and database handles[6].

3.1.2 Web services

SOAP Web services can be imported into the platform and can be used in much the same way as native widgets. Also in communication among Web services, the platform arbitrates in their

[5] http://docs.python.org/2/library/pickle.html

[6] A handle in computer programming means an abstract reference used for accessing resources such as files, databases and alike.

communication and performs intermediate data serialization (pickling), so the same requirements regarding interchanged data types apply as in native widgets described in Section 3.1.1. However, there is one exception: in case of data streams, the exchanged data is not serialized and can be of any kind, as long as the Web services are synchronized regarding its type and format.

REST Web services do not come with formalized definition of their interface. They will be supported by a dedicated widget that will enable the user to define the parameters of the HTTP POST request for a particular REST service. Inclusion of REST Web services into the platform will cause no additional requirements or restrictions on data types or formats, since all communications and operations of REST Web services are done using HTTP protocol and the whole HTTP response object in Python is picklable.

3.2 Data and model representation

The main data representation language used in the collaborative system of Lrn2Cre8 is agreed to be the RDF (Resource Description Framework). RDF is designed for machine readability, preservation of semantics and information exchange among applications, thus it is well suited for representing data and models that are exchanged among Web services.

RDF can be roughly described as data representation in which:

- Data is represented as a collection of Subject—Predicate—Object triples. Subject and object stand for two things in the world, while the predicate stands for a relation among them.
- Each of the three elements in a triple is a name of *global entity*, a name of *local entity* or a *text literal*.

Names refer to things in the world and are given as URIs (Uniform Resource Identifiers) [9]. Names of global entities are shared among many RDF documents, so they always have the same meaning. The names of local entities are referred to only within the document where they reside. Text literals represent raw data, like numbers, raw text etc.

The reader wanting to know more about RDF representation, including the graph representation of RDF triples collections can find more information in [14].

3.2.1 RDF notation

There are two main notations used for writing RDF:

- RDF/XML and
- Turtle (pure RDF subset of Notation 3, which can also express logic).

In Lrn2Cre8, we suggest using the Turtle notation, since it is less verbose, much easier to read and write by humans and is currently the most widely used one. A common argument for RDF/XML used to be that it is standardized by W3C, which the Turtle was not, but this is no longer the case, as Turtle is getting standardized by W3C as well[7].

Turtle and RDF/XML are merely notational formats and use of one or the other does not affect the RDF data representation. Therefore, conversion from one format to the other is not difficult and there are several free tools for it. All the already existing data resources, which the partners might have in RDF/XML are thus also useful.

[7] http://www.w3.org/TR/turtle/

3.2.2 Data types

Elements of RDF triples are global or local names, or text literals. For text literals we can define also their data types (in the sense of computer programming, not for example in the sense of types in ontologies). This is done by appending two carets (^^) and a name of the datatype to the literal. A common convention is to use datatypes of the XML Schema and this will be also the convention in RDF of Lrn2Cre8. In this way, each data type can be uniquely referred to with a URI, for example like this:

```
"0"^^<http://www.w3.org/2001/XMLSchema#boolean>
```

for a zero that is of a boolean data type.

3.2.3 Lrn2Cre8-specific URIs

Use of URIs is an essential characteristic of the RDF data representation. URIs of RDF data in Lrn2Cre8 will come from common core vocabularies, such as the RDF[8], Dublin Core[9] and FOAF[10]. However, it is expected that for some of the data we will need to provide new, project specific URIs. For this reason it is useful to have a common URI schema for the project.

We propose the use of project's URL as the first part or the base of project specific URIs. Second part should follow a shallow hierarchy that splits resources as *data*, *model* and *system* specific concepts, with hash (#) prefixed items as the final part of the URI. Each project specific URI would thus follow the template:

```
<base><hiearchy>#<item>
```

where <base> is *http://lrn2cre8.eu/*, the <hierarchy> is defined as:

- <base>
  - <base>/resource/
    - <resource>/data#
    - <resource>/model#
    - <resource>/system#

and the <item> is a unique (unique to the project's namespace) name of a concept that we want to represent with a given URI.

A project's specific understanding of a concept might be represented as:

```
<http://lrn2cre8.eu/resource/model#concept>.
```

The proposed schema is, of course, subject to possible changes and adaptations. It is not possible to envisage what kind of concepts, data and models will arise from the project and what kind of representation needs might occur in future Lrn2Cre8 developments. The proposed schema, particularly its hierarchy, should thus be perceived as a recommendation rather than a specification.

Providing *look-up* URIs are identifiers and in general they do not provide any information about the resource they identify. For example, although a URI such as `http://someproject.net/London/C02/sensor42#2013october1` might suggest that it represents a CO_2 sensor reading in London on a particular day, it could in fact identify a recipe for a particularly delicious cake in

[8] RDF namespace: http://www.w3.org/1999/02/22-rdf-syntax-ns#
[9] Dublin Core namespace: http://purl.org/dc/elements/1.1/
[10] FOAF namespace: namespace: http://xmlns.com/foaf/0.1/

somebody's collection of baking recipes. The URIs should not be screened for information about the referenced resources as this is not their role: they are only identifiers.

It is thus a commonly accepted good practice to enable retrieval (*look-up*) of descriptions of the resources that the URIs identify. This can be done in a straightforward way for the URIs of the *http:* schema by providing HTML description for humans and RDF data for machines that access the URI through the HTTP protocol. In case of our exemplary project's specific representation of a concept, when accessing the URI `http://lrn2cre8.eu/resource/model#concept` the HTTP protocol removes the # symbol and everything after it, so we receive the HTML description that is available at `http://lrn2cre8.eu/resource/model`. In this case a description of all resources that deal with models, including a description of the *concept*. The same address could serve the RDF representation of the *concept*, if a machine would access it. Or a PNG image or MP3 audio file, etc. Explanations of how this can be implemented and some other URI recommendations are available in W3C documentation [12]. For the URIs of Lrn2Cre8, we propose mandatory availability of URI descriptions, not constraining the implementational details.

3.3 Fallback representations

In case the preferred RDF data representation proves to be too difficult or costly to use or manage in a specific Lrn2Cre8 dataset, another formalized representation should be used. The Lrn2Cre8 data specification supports any serializable data format as explained in Section 3.1.1. For such cases, the XML should be used as the first fallback format of choice.

Other data formats (like JSON[11], YAML[12], etc.) can be used if agreed upon and if their use makes a specific benefit for the given Lrn2Cre8 application.

Free and ad hoc formatted strings can be used and are supported, but should be limited only to specific components and situations (early prototypes).

4 Summary of specifications

Here we summarize all the specifications and recommendations for the components of Lrn2Cre8's outputs that are unified into a collaborative technical infrastructure.

Specifications:

- Software components are implemented as Web services or Python widgets.
- Web services must follow either SOAP or REST protocol.
- We aim at RDF representation of our data.
- Datatypes of RDF literals are datatypes of the XML Schema.
- Project specific data must follow the URI schema as defined in Section 3.2.3.
- URIs designed in Lrn2Cre8 must provide at least HTML descriptions of the resources that they represent.

Recommendations:

- RDF data is serialized according to the Turtle (N3 subset) format.
- In case the RDF framework is too difficult to follow for representation of a certain dataset, the XML format should be used.

[11] http://www.json.org/
[12] http://www.yaml.org/

Appendices

A Lrn2Cre8 workflow component specification

The following information must be provided for each workflow software component. This is the same requirement as for the ConCreTe working specification.

Note: not all details are required for all types of integration but should be provided for the sake of completeness.

1. **Contributor**: name of the institution that is providing the component.
2. **Contact**: name and contact of the person responsible for the component.
3. **Name**: name of the component.
4. **Description**: description of the component in form of a short free text.
5. **References**: references of the component. If the component is published in a scientific publication, a citation of the original work.
6. **Licence**: the licence under which the component is available (open source, proprietary, non-commercial, ...).
7. **System requirements**
   - **Hardware requirements**: processing, memory, and storage requirements of the components.
   - **Operating system**: operating system under which the component can run.
8. **Implementation details**
   - **Language**: programming language in which the component is written.
   - **Third-party libraries**: whether the component relies on some third-party libraries. If yes, a list of the libraries on which it relies and their respective licences.
9. **Integration**: specify which type of integration is most suitable for the component. Choose one of the following:
   - native integration with Python (C, C++, Python)
   - tight integration using wrappers (Java, C#)
   - integration of standalone command-line executables
   - integration with web services
     - SOAP
     - REST
   - web integration (remotely hosted software, only for visualizations)
10. **Integration issues**: the issues that the contributor of the component foresees may arise when integrating this component into ClowdFlows. Examples include issues like thread safety and statefulness.
11. **Inputs**: number of data inputs and their name, type, and format. Number input parameters to be controlled from the workflow and their name and type.
12. **Outputs**: number of outputs and their name, type, and format.

B Component specification example

1. **Contributor**: Queen Mary University of London
2. **Contact**: Geraint Wiggins
3. **Name**: Information Dynamics Predictor
4. **Description**: The component provides a distribution over the alphabet of a multidimensional smoothed, interpolated mixed-order Markov Model
5. **References**: Pearce, M. T. (2005). PhD Thesis, City University, London. [11]
6. **Licence**: open source.
7. **System requirements**
   - **Hardware requirements**: : CPU 2GHz, 2 GB RAM, 1 GB disk space
   - **Operating system**: Unix
8. **Implementation details**
   - **Language**: C
   - **Third-party libraries**: open-source, non-commercial libraries and lexical resources
9. **Integration**: Via RESTful Web services
10. **Integration issues**: none foreseen
11. **Inputs**: RDF specification of model data
12. **Outputs**: Distribution over alphabet of model.

References

[1] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: The Konstanz Information Miner. In Christine Preisach, Hans Burkhardt, Lars Schmidt-Thieme, and Reinhold Decker, editors, *Data Analysis, Machine Learning and Applications*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 319–326. Springer Berlin Heidelberg, 2008.

[2] Janez Demšar, Blaž Zupan, Gregor Leban, and Tomaz Curk. Orange: From experimental machine learning to interactive data mining. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, volume 3202 of *Lecture Notes in Computer Science*, pages 537–539. Springer Berlin Heidelberg, 2004.

[3] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[4] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[5] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R. Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.

[6] Graham Klyne, Jeremy J Carroll, and Brian McBride. Resource description framework (RDF): Concepts and abstract syntax. *W3C recommendation*, 10, 2004. http://www.w3.org/TR/rdf-concepts/ (accessed on 5th of January 2014).

[7] Janez Kranjc, Vid Podpečan, and Nada Lavrač. ClowdFlows: A cloud based scientific workflow platform. In Peter A. Flach, Tijl Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524 of *Lecture Notes in Computer Science*, pages 816–819. Springer Berlin Heidelberg, 2012.

[8] S. Majithia, M. Shields, I. Taylor, and I. Wang. Triana: a graphical web service composition and execution toolkit. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 514–521, 2004.

[9] Larry Masinter, Tim Berners-Lee, and Roy T Fielding. Uniform resource identifier (URI): Generic syntax. 2005. http://tools.ietf.org/html/rfc3986 (accessed on 5th of January 2014).

[10] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–940, New York, NY, USA, 2006. ACM.

[11] Marcus. T. Pearce. *The Construction and Evaluation of Statistical Models of Melodic Structure in Music Perception and Composition*. PhD thesis, Department of Computing, City University, London, UK, 2005.

[12] Leo Sauermann, Richard Cyganiak, and Max Völkel. Cool uris for the semantic web. 2011. http://www.w3.org/TR/cooluris/ (accessed on 5th of January 2014).

[13] A. Smaill, G. A. Wiggins, and E. Miranda. Music representation – between the musician and the computer. In M. Smith, G. Wiggins, and A. Smaill, editors, *Music Education: An Artificial Intelligence Perspective*, pages 108–119. Springer, London, 1993.

[14] Joshua Tauberer. What is RDF and what is it good for? http://rdfabout.com/intro/ (accessed on 5th of January 2014).

[15] G. A. Wiggins. Computer-representation of music in the research environment. In T. T. Crawford and L. Gibson, editors, *Modern Methods for Musicology: Prospects, Proposals and Realities*, Digital Research in the Arts and Humanities, pages 7–22. Ashgate, Aldershot, UK, 2009.

[16] G. A. Wiggins, M. Harris, and A. Smaill. Representing music for analysis and composition. In M. Balaban, K. Ebcioğlu, O. Laske, C. Lischka, and L. Soriso, editors, *Proceedings of the Second Workshop on AI and Music*, pages 63–71, Menlo Park, CA, 1989. AAAI.