Crowd-powered recommendation for continuous digital media access and exchange in social networks

FP7-610594

D2.2

First Reference Framework Release and Evaluation Report

Dissemination level:	Public
Contractual date of delivery:	Month 12, 30 September 2014
Actual date of delivery:	Month 12, 30 September 2014
Workpackage:	WP2. Requirements and Reference Framework
Task:	T2.2 Evaluation metrics, ground truth and data T2.3 Reference framework implementation T2.4 Reference framework evaluation
Type:	PMB Final Draft
Approval Status:	Draft
Version:	1.0
Number of pages:	40
Filename:	CrowdRec_WP2_D2.2_MOV_30092014_V1.0.pdf

Abstract

This deliverable reports the results achieved in WP2 “Requirements and Reference Framework” about the first release of the reference framework and its evaluation. The reference framework, publicly released with the name **Idomaar**, has a four-component architecture that makes it possible to experiment with new, as well as existing, recommendation algorithms in an architecture-independent environment that supports the assurance of the consistency and reproducibility of results.

A basic implementation of the four components has been released. Next iterations will expand each components either more options or more features. This deliverable also contains related information on data sets and independent releases of algorithm implementations.
The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	2014.07.14	Created template	Roberto Turrin
0.2	2014.08.04	Updated document structure	Roberto Turrin
0.3	2014.08.07	Added main descriptions to sections	Roberto Turrin
0.4	2014.09.04	Completed most sections	Roberto Turrin
0.5	2014.09.05	Added provisioning section	Davide Malagoli
0.6	2014.09.24	Added evaluation section	Domonkos Tikk
0.7	2014.09.24	Added evaluation metrics and data sets	A. Lommatzsch
0.8	2014.09.24	Added RiVal	Alan Said
0.85	2014.09.25	Finalized document draft	Roberto Turrin
0.9	2014.09.28	Revision	Martha Larson
0.95	2014.09.29	Finalized document draft, second review	Roberto Turrin
1.0	2014.09.30	Final formatting	Martha Larson

Author list

Organization	Name	Contact Information
MOV	Roberto Turrin	roberto.turrin@moviri.com
TUD	Babak Loni	b.Loni@tudelft.nl
MOV	Davide Malagoli	davide.malagoli@moviri.com
TUB	Andreas Lommatzsch	andreas@dai-lab.de
MOV	Andrea Condorelli	andrea.condorelli@moviri.com
GRA	Viktor Gal	viktor.gal@gravityrd.com
GRA	Domonkos Tikk	domonkos.tikk@gravityrd.com
TUD	Mark Melenhorst	m.s.melenhorst@tudelft.nl
TUD	Alan Said	alansaid@acm.org
TID	Alexandros Karatzoglou	alex@tid.es

Executive Summary

This deliverable describes the first release of the reference framework and its evaluation at the end of the first year (M12) of activities of "**WP2: Requirements and Reference Framework**". The deliverable contributes to *Milestone 1 "First Reference Framework Release and Basic Real-world Deployment"*.

The requirements collected in D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 8, page 41) have been taken into consideration to define the set of properties the reference framework has to fulfill. The Reference Framework - named **Idomaar** - makes it possible to run offline tests, providing communication and data interfaces to decouple the evaluation logic from the recommendation service, enabling both open source and proprietary software to be impartially compared. Starting from these requirements, this deliverable developed the following properties the reference framework has to implement:

- grant consistency and reproducibility of results
- architecture independence
- standardization.

Furthermore, the reference framework must allow the implementation and execution of a selection of the recommendation algorithms developed in **"WP3: Stream Recommendation algorithms"**, in particular taking into consideration the possible streams of data and the RICHeS challenges, i.e., real-time management of **I**nteraction, **C**ontextual, **H**eterogenous, and **S**ocial data sources.

As initially described in D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 8, page 41), Idomaar consists of four main components:

- **Data container**, where datasets are stored. Currently, we released it with the MovieTweetings dataset.
- **Computing environment**, where the recommendation algorithms are executed. The first release of Idomaar contains the configuration for WrapRec as execution environment. Consequently, all algorithms implemented in WrapRec are available within Idomaar. these include two algorithms addresses the RICHeS challenges of Context and Interaction: "Cross-Domain Collaborative Filtering with Factorization Machines" and "'*Free Lunch' Enhancement for CF with Factorization Machines"* algorithm described in *D3.1 "Scalable Stream Recommendation algorithms"* (Section 3.3.1, page 14). The computing environment is designed such that a practitioner can configure their custom computing environment to run their implemented algorithms. Additional algorithms will be chosen for future implementations on the basis of CrowdRec requirements, the next version of which are due to be released in M20 as D2.3.
- **Evaluator**, the strategies and metrics to evaluate and compare the recommendation algorithms. Idomaar integrates RiVal and consequently allows experimenting with all

policies and measures already implemented in the RiVal framework. Furthermore, custom solutions can be implemented relying on the RiVal programming interfaces.

- **Orchestrator**, the coordinator of the whole reference framework. Idomaar currently implements a simple Python orchestrator.

The first release of the Reference Framework provides a *proof-of-concept* of its effectiveness and feasibility, providing a basic working implementation of all the required components. Future work will involve extending the number of released options (e.g., the pre-configured, ready-to-use computing environments) is still limited and extending the current deployment of some of the components (e.g., the orchestrator).

Next releases of the Reference Framework will target the evaluation of stream of data and expand all the four components, by both increasing the number of implemented options - e.g., a larger selection of algorithms from WP3, together with the required computing environments - and extending their capabilities - e.g., the evaluator to allow both off-line and on-line testing. As mentioned above, specific decisions about algorithm selection, and other priorities for idomaar are made on the basis of CrowdRec requirements, which are iteratively defined over the course of the project.

In addition to discussing the Reference Framework, this deliverable contains information about related develops that have occurred in WP2, including datasets, (Task 2.2) and also implementations of CrowdRec algorithms that were released as independent releases accompanying publications (Task 2.3).

Acronyms

SaaS	Software as a Service
RF	Reference Framework
CPE	Computing Environment (part of the Reference Framework)
ORC	Orchestrator (part of the Reference Framework)
EVL	Evaluator (part of the Reference Framework)

Table of Contents

1. Introduction

This deliverable describes the main results achieved in WP2 "Requirements and Reference Framework" during the first year, in particular focusing on the **first reference framework release and evaluation report**. The reference framework implementation (*Task T2.3 "Reference Framework Implementation"*) has been driven by the requirements elicited from the social network and SME partners and collected in *Deliverable D2.1 "First Iteration Requirements plus Evaluation Specifications"*.

A release of the CrowdRec reference framework involves elements related to multiple tasks in CrowdRec WP2 and WP3:

- The datasets available to practitioners to run experiments, as decided in *Task T2.2 "Evaluation metrics, ground truth and data"*.
- The evaluation strategies and metrics, as defined *Task T2.2 "Evaluation metrics, ground truth and data"*.
- A selection of the recommendation algorithms developed in *WP3 "Stream Recommendation Algorithms"*
- A modular software environment to run experiments, implemented in *Task T2.3 "Reference Framework Implementation"*. This allows executing and evaluating the implemented recommendation algorithms using the available datasets and evaluation strategies and metrics.

The reference framework will be evaluated in *Task T2.4 "Reference framework evaluation"* to understand its suitability for evaluating recommender system algorithms on the basis of the CrowdRec 3-D evaluation model. This suitability is the pre-requisite for applying the Reference Framework for evaluation.

At Milestone 1, at the end of the first year (M12), we released the First Release of the reference framework with all basic components and modules required for a walkthrough of the functionalities.

The development of the first release of the reference framework required a complex process. In fact, after analyzing the existing initiatives (see Section 3), we started from the requirements resulting from Task T2.1 (and reported in Deliverable D2.1) and we refined and detailed the features and properties that the reference framework has to be based on (see Section 4.1). Once requirements have been defined, we compared and discussed its architecture (see Section 4.2), coming out, in the end, with a final proposal (see Section 4.3).

Subsequently, we implemented an initial proof-of-concept was implemented. In order to assess whether the proof-of-concepts fulfilled the requirements of the CrowdRec consortium, a workshop meeting was held during June 2013 at Moviri in Milan. At this meeting a pre-release

was shared with all the partners in the CrowdRec consortium. The partners were asked to run the Reference Framework on their own systems, and attempt to carry out integration and evaluation tasks typical of those which the Reference Framework must ultimately support.

The result of this meeting was a compile list of feedback and issues that were used to improve the Reference Framework specifications, and guide development towards the release at M12 Milestone 1. At this meeting, we also selected recommendation algorithms developed within RecSys and addressing the RICHeS challenges to be made available in the first release of the reference framework. Specifically, these algorithms are *"Cross-Domain Collaborative Filtering with Factorization Machines"* and "'*Free Lunch' Enhancement for CF with Factorization Machines"* algorithm described in *D3.1 "Scalable Stream Recommendation algorithms"* (Section 3.2.2 and 3.3.1). The first release was made available on the GitHub repository at the end of September 2013. At the time of this release, the CrowdRec consortium decided to name the reference framework Idomaar, which is a Hungarian word meaning "trainer" and evoking control and coordination of all the elements of recommender system evaluation. The consortium uses the designations "Idomaar" and "CrowdRec Reference Framework" interchangeably.

At the project level, CrowdRec strives to release reference implementations of algorithms with the scientific papers that it publishes. The releases are often made as accompanying material for papers arising from work within the algorithms workpackages WP3 and WP4. They may not, however, be necessarily integrated into the Reference Framework. In the first project year, the algorithm *"Gaussian Process Factorization Machines for Context-Aware Recommendation"* (D3.1 Section 3.1.3) was released as an independent implementation http://trungngv.github.io/gpfm. Other algorithms earmarked for possible release in Year 2 are Context-Aware Recommendation by Learning Context Representation (D3.1 Section 3.2.1) and RankSLDA (D4.1 *"Crowd Engagement Algorithms*" Section 3.2). Integration depends on the resource prioritization determined by recommender system requirements from Task 2.1.

In this deliverable, we go beyond reference implementations and the Reference Framework to cover information from other related WP2 tasks. This information is pertinent to the current release, but also to future releases of the Reference Framework. Here we describe each of the tasks covered in turn.

T2.2 - Evaluation metrics, ground truth and data

The task defines the strategies and metrics to be used to evaluate the recommender algorithms. As set out in D2.1, CrowdRec evaluation follows the 3D-model, which conceptualizes the performance of a recommender system as being related to three dimensions: recommendation quality, technical requirements as well as business-oriented aspects. In this task, we analyze metrics and evaluation approaches in order to inform our decisions on which techniques should be implemented within the reference framework. In addition, this task develops datasets necessary for evaluating CrowdRec algorithms.

T2.3 - Reference framework implementation

The task includes to the design and implementation of the overall Reference Framework,

whose goal is allowing testing a set of recommendation algorithms (as proposed in WP3) before being deployed in real-world scenarios for user tests. The task also includes implementation of algorithms developed within CrowdRec. Some of these algorithms are selected for integration within the Reference Framework, and others are released as independent implementations. The decision is based on the requirements for evaluating individual algorithms, which are formulated by Task 2.1. Testing is defined according to the evaluation strategies and metrics presented by Task T2.2 - Evaluation metrics, ground truth and data, providing tools to compute the related performance of the algorithms.

The reference framework provides (i) the environments to execute, experiment, and test recommendation algorithms, (ii) a set of implemented recommendation algorithms, and (iii) the data set to use.

At the end of the first year, the first release of the Reference Framework is composed by the key components required to prove its feasibility and to execute a first run of experiments with a few base algorithms and a selected dataset.

The task will continue during the rest of the project, enlarging the base of recommendation algorithms with an extract of the ones proposed in WP3, integrating further evaluation strategies and metrics, and including new datasets.

T2.4 - Reference framework evaluation

The task refers to the evaluation of the algorithms that are implemented in the reference framework in Task T2.3, testing their performance and suitability for deployment in large-scale social networks. The evaluation of such algorithms firstly requires to evaluate the Reference Framework itself - specifically, whether or not it is able to support the CrowdRec 3D evaluation model - as described in D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 6, page 31). This task started in M10; at the end of first year, we evaluated the framework from three aspects: (i) fulfillment of the requirements, (ii) usability and technical integrity, and (iii) business applicability. The task will proceed during the rest of the project to constantly provide assurance for the suitability of the reference framework for the eventual deployment in large-scale environments.

1.1. Document outline

The rest of the document is organized as follows. Section 2 describes the public datasets candidate to be included in the reference framework. Section 3 discusses the requirements of the reference framework and proposes a solution, detailing the distinguishing points with respect to existing options. Section 4 describes the architecture of the reference framework and the four components it is composed of. Section 5 reports the formal specification of data model, output format, and communication protocol. In Section 6 we describe the tools used to automatically provision the reference framework components. Section 7 presents what the first release of the reference framework includes, in terms of computing environment and

algorithms, evaluator, and datasets. The dissemination activities of such release are reported in Section 8. Section 9 evaluates the current version of the reference framework. Finally, we draw some conclusions and plan the activities of next iterations in Section 10.

Appendix A reports the existing initiatives in the settings of recommendation frameworks.

2. Datasets

The Reference Framework is envisaged to be accompanied by datasets supporting experimentation with recommendation algorithms. For the scientific evaluation we will focus on publicly available datasets. This ensures that the results can be easily compared with results from other research groups and that the developed algorithms go beyond state-of-the-art algorithms. The first release of the Reference Framework was accompanied by a version of the Movie Tweetings data set whose purpose was to demonstrate the CrowdRec data model. Note that an another version of dataset is used in the RecSys 2014 challenge (http://recsys.acm.org/recsys14/challenge/) [2]

In this section, we describe several other datasets that have been developed in the project, and fulfill the specifications in Deliverable D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 7, page 35) that identified the properties datasets should fulfill to be useful for CrowdRec.

News recommendations

The CLEF-NewsREEL-2014 dataset (http://www.clef-newsreel.org/dataset/) describes the interaction of users with articles. The dataset contains timestamp describing the stream of user-item interaction. For items as well for the users detailed meta-data is available.

The dataset is relevant for the CrowdRec scenario because it provides heterogeneous stream-based data fitting well the RICHeS criteria.

Tuenti Social Network Dataset

During the first year of the project, Tuenti shared social network data with TID. The data was shared in accordance with CrowdRec "Privacy and informed consent"
procedures (DoW B4 Section 4.1.) Sharing with the rest of the consortium has been planned for Year 2. The data shared by Tuenti includes:

- Social graph: a fully anonymized user to user connection dataset along with the timestamp of the formation of the connection
- Wall to wall comments: a fully anonymized, communication graph between Tuenti users (does not include the actual messages)
- Photo tags: the photos tagged by users
- Ads: ad impressions and clicks by users along with ad features and user demographic characteristics (fully anonymized).
- Videos: videos viewed by users, portions of videos viewed anonymized

Thus far, the Tuenti data has been used for effort related to WP5. However, we anticipate that it will be an important for design and development of algorithms moving forward.

SoundCloud dataset

Deliverable D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 7.1.4, page 39) identified the Sound Cloud comments dataset as one of candidate datasets fulfilling the properties identified in D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 7.1.2, page 37). The dataset is also mentioned in D5.1 in regards to the SoundCloud use scenario.

The Sound Cloud dataset is set of "**sounds**" from the Sound Cloud website (https://soundcloud.com/) that were accessed via the SoundCloud API. SoundCloud provided an exhaustive list of all the Creative Commons sounds that was published to the site in the years 2012 and 2013.
Every sound is composed by only one track and several comments. Tracks, comments, and users have some attributes (e.g., track has the attribute "genre", user has the attribute "username", and comment has the attribute "created_at").

The retrieved dataset is **composed by** about 2M comments on about 300K different tracks: only about 35K tracks has more than 10 different commenters.

We have analysed both properties about single "items" such as the average of commenters per track and properties related to couple of items such as the number of co-commenter (i.e., we denote as **co-commenters** the users that have commented at least one common track). Briefly:

- Each track has 2.9 comments on average (considering both commented and uncommented tracks).
- Each commented track has 9.1 comments on average.
- Every (commented) track is commented by 5.8 distinct commenters on average.
- Every user comments 2.5 distinct tracks on average.
- Every author comments his/her own track 4.6 times, on average.

Additional information is available through SoundCloud public API's: user groups, user playlists, follower/followee relations. A sample of such data has been retrieved in order to verify its content.

3. Reference framework

The reference framework is an implementation of algorithms that have been developed within CrowdRec and serves to allow the testing of these algorithms before they are deployed in the real-world social networks for large-scale user tests. The reference framework is also made available to the research community.

3.1. Requirements

The requirements for the first release of the Reference Framework were jointly established by the vendors (Gravity and Moviri) and the academic partners (TUB, TUD) in the consortium (cf. D2.1 “First Iteration Requirements plus Evaluation Specifications”, p. 41). The most basic requirement of the Reference Framework is that it is able to evaluate the ability of CrowdRec algorithms to meet the recommender system requirements that arise in Task 2.1 “Requirements”. In this sense, the Reference Framework requirements can be considered meta-requirements. The requirements for recommender system algorithms that most strongly influence the requirements for the reference framework are those involving speed, incremental updates and dynamic contexts.

In order for the Reference Framework to be useful, the vendors developed a set of requirements that would enable them to make the best possible use of the Framework. These requirements were taken to be representatives of the needs of the targeted end users for the framework:

- **Consistency and reproducibility**: a consistent environment to run and evaluate quality, performance and scalability of algorithms, granting the reproducibility of experiments and related results.
  In fact, actors (e.g., practitioners, researchers,) in the fields of recommender systems face with several issues when using existing recommendation framework, such as:
- The implementation of testing is different according to the framework; note that even though claiming the same splitting technique (e.g., leave-one-out), minor differences can lead to different results.
- Starting from the pseudo-code (e.g., as published in a paper) the implementation can slightly differ according to the recommendation solution. For instance, a different initialization of data can lead to different output.
- Some algorithms can have completely different performance (in terms of execution time and required memory) on different frameworks, due to - as an example - different data model and access.
- **Architecture independent**: the framework must not depend on specific architectures, but it has to provide common specification to allow all consortium members (and further practitioners) to implement their algorithms in their preferred architecture/language
- **Standardization**: let algorithms run on same data, managing also streams of data

Together with the meta-requirements mentioned above, these requirements represent the ideal characteristics for the Reference Framework, and informed the development of the overall specifications. One particularly important aspect was the decision between a full and a minimal framework, which was studied in depth by the consortium. The considerations that informed this decision are discussed in more detail in the next subsection.

3.2. Background

Before starting the design of the reference framework, we deeply analyzed and evaluated the existing initiatives in the field of recommender systems and data mining, with the main objectives of avoiding to duplicate something already existing, being aware of the capabilities of the existing solutions and how they pose with respect to the CrowdRec reference framework’s requirements (c.f. Section 3.1).

See Appendix A for a detailed description of the identified initiatives.

3.3. Full vs. minimal framework

We compared two opposite solutions for the reference framework implementation - either a full framework or a minimal framework in order to understand the advantages and disadvantages of the two approaches.

The **full framework** provides interfaces (a common language and libraries) to access data and implement the recommendation algorithms. On the other hand, the **minimal framework** only provides a common output format (procedure and documentation) that the implemented algorithms have to respect.

The following table compares the two solutions in terms of: quality of the implementation, reproducibility and comparison of results, effort to define and implement the reference framework, effort to implement the algorithms in WP3, scalability control, and data modeling. Green or red background indicate, respectively, when a solution is better or worse than the other one.

Property	Full framework	Minimal framework
implementation quality	homogeneous and controlled development and evaluation environment	low quality (no control on implementation)
reproducibility and comparison of results	any user can run any implemented algorithm, reproduce the same tests and compare results	lack of an execution framework to reproduce experiments
reference framework definition: effort	high effort to define: (i) data model, (ii) algorithm interfaces/apis, (iii) evaluation interfaces/apis	minimal effort to define output format
algorithm implementation: effort	high effort: it might require to re-develop existing algorithms according to the selected language	low effort: it is possible to re-use existing implementations
scalability control	we can grant and evaluate/measure scalability. Minimal effort required if the algorithms respect the framework specification	hard to grand and evaluate/measure. high/unpredictable effort: engineering an algorithm might not be straightforward
data modeling	It is provided by the framework and allows to manage data stream (big data)	Not commonly defined. It might be not possible to manage streams of data (10GB/day). Data ingestion might be challenging

The main outcomes of the comparison show that the full framework requires a higher effort for its bootstrap, but has a number of advantages in terms of quality, reproducibility of results, scalability control, and data modeling.

At the end of this analysis - according also to the requirements specified in Section 3.1 - we agreed for an implementation of the reference framework in the middle between full and minimal, as described in the following paragraphs.

3.4. Proposed framework

The reference framework is based on automatically-provisioned **virtual machines** (VMs) to support easy use, as described in D2.1 “First Iteration Requirements plus Evaluation Specifications” (Section 8, page 41). This section represents a review and extension of the material provided in D2.1. With regard to the use of virtual machines:

- Each VM can be based on one or more existing frameworks (e.g., on Apache Mahout or proprietary solutions). The use of VM makes practitioners free to use the technology they are more confident working with.
- no constrained to a specific language (e.g., Java for Lenskit).
- no constrained to a given data model (e.g., DB, graph, csv).

In this deliverable, the usage of self-contained VMs has been further analyzed and experimented. This choice respects the requirements and properties discussed in Section 3.1, granting **evaluation to be consistent** among different algorithms and ensuring **reproducibility** of results. Furthermore, the approach facilitates **testing** and **prototyping**:

- **testing**. E.g., A researcher has his own algo already implemented, he creates the configuration for the virtual machine with all required packages, and runs the orchestrator to get all metrics (e.g., recall, execution time, consumed memory,...) on an existing dataset
- **prototyping**. E.g., a new algorithm is designed, the researcher can choose the language he prefers to implement the algorithm, he develops the solution and run the evaluation to compare it with all the other existing algorithms on the reference framework.

3.5. Distinguishing properties

The reference framework proposed within CrowdRec differentiates from the existing recommendation solutions and tools (see Appendix A) in different points:

- It is architecture and language independent, granting a fair comparison of
    - different algorithms, whatever framework/technology/language they are based on
    - different implementations of the same algorithm on different technology solutions (e.g., Cosine implemented on Mahout vs. the Cosine implemented on Lenskit)
- It allows managing stream dataset
- It allows monitoring

- computing times for each stage of the recommendation workflow (e.g., data ingestion, training data processing, recommendation).
- machine cpu/memory consumption (instrument at system-level)

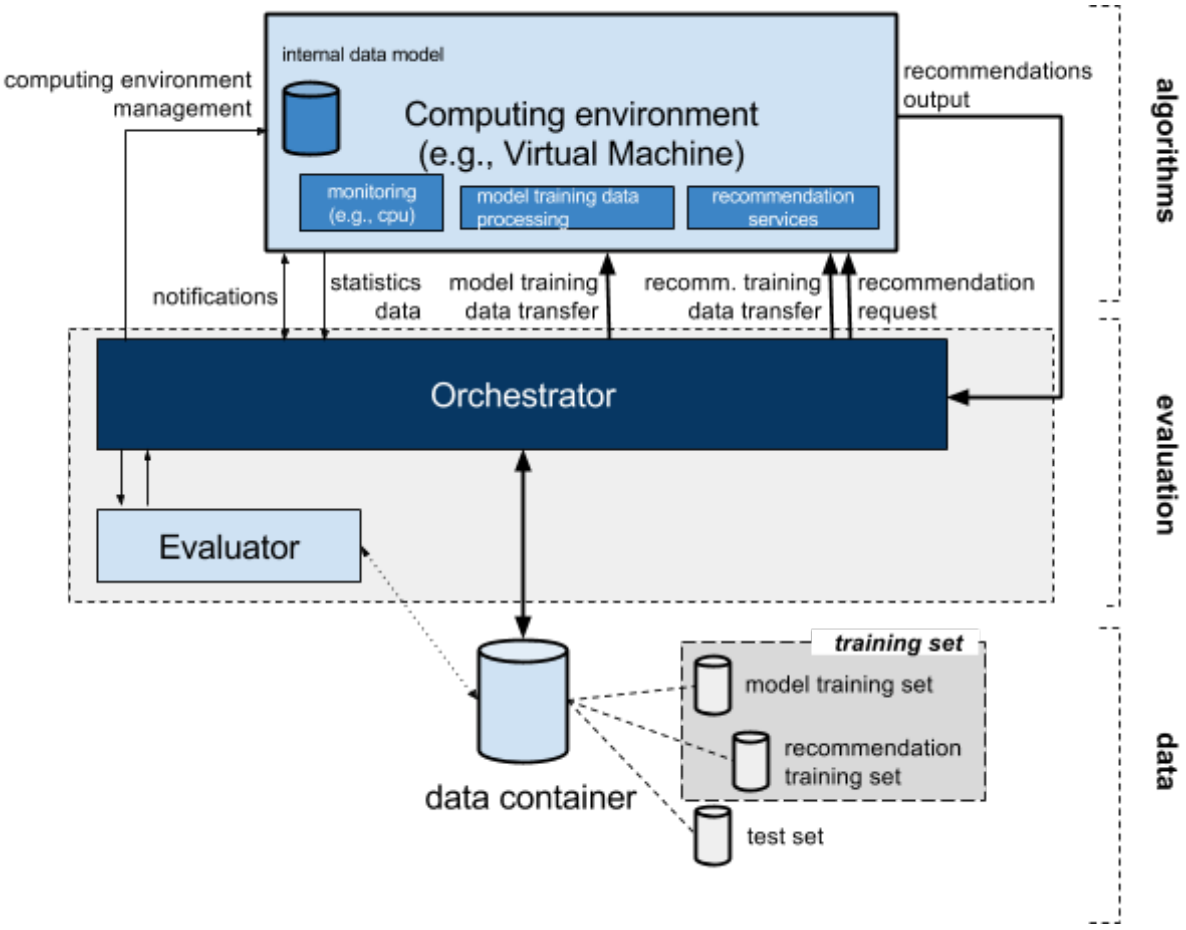
The aforementioned properties, together with the requirements presented in Section 3.1, have driven the design of the reference framework architecture, described in the following sections.

4.Reference framework architecture

The high-level overview of the reference framework architecture was already presented in D2.1 “First Iteration Requirements plus Evaluation Specifications” (Section 8, page 41). In this section, we go through each single component providing more details.
The reference framework is formed by three layers:

- the **algorithms** to test, both state-of-the-art algorithms and new solutions implemented within the CrowdRec project, e.g., the algorithms developed in WP4 *“Crowd Engagement Algorithms”*.
- the **evaluation logic**, experimenting with the available algorithms in order to compute both quality (e.g., RMSE, recall) and system (e.g., execution and response time) metrics. The framework will include some evaluation policies, free to be extended.
- the **data**, i.e., the datasets made available to the practitioners (e.g., the MovieTweetings).

Algorithms, evaluation logic, and data will be as decoupled as possible to allow to experiment with most existing solutions - no matter the technology they are implemented on - granting reproducible and consistent comparisons.



The 3 layers are implemented by 4 modules, referred to as reference framework components: the data container (DC), the evaluator (EVL), the orchestrator (ORC), and the computing environment (CPE).

4.1. Data Container

The data container contains all datasets available in the reference framework. The dataset can be split into three parts according to the evaluation strategy implemented in the evaluator: the model training set, the recommendation training set, and the test sets.

Data must be conform to the data model format specified in Section 5.1.

We started exploring possible datasets to be included into the reference framework to be available to practitioners to experiment with their recommendation solutions. See Section 2 for the example of datasets.

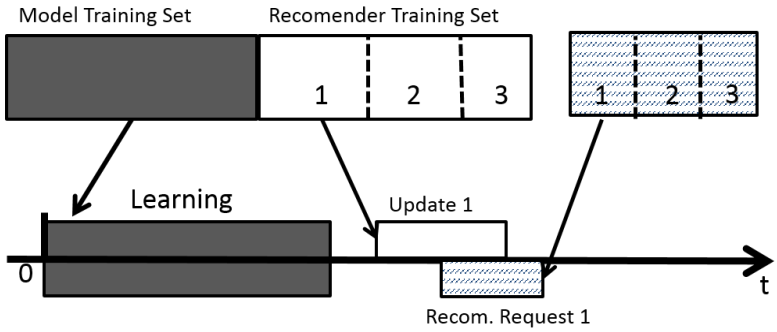
4.2. Evaluator

The reference framework focuses detailed, reproducible evaluation of different recommender algorithms based on an exactly defined environment. The reference framework supports offline evaluation of static as well as stream-based datasets. Dataset splits (into training and test set) can be done either randomly or based on predefined training and test folds.

The evaluation framework provides a comprehensive tool for the detailed, reproducible benchmarking of recommender algorithm. It covers the academic as well as the business-oriented aspects. The framework is highly modular and open for additional metrics and evaluation approaches.

The evaluator contains the logic to (i) split the dataset according to the evaluation strategy and (ii) compute the quality metrics on the results returned by the recommendation algorithm, as the ones described in D2.1 "*First Iteration Requirements plus Evaluation Specifications*" (Section 6.3, page 33).

The EVL divides the dataset into training and test set (Figure 4).



The **test set** contains the data completely hidden to the recommendation algorithm. These data are visible only to the evaluator (and, of course, the orchestrator), while the computing environment is totally unaware of such data.

The **training set** contains the data visible to the computing environment and can be used to train the recommendation algorithms. In addition, we introduced a more advanced logic with respect to the common training/test set splitting pattern. In fact, we distinguish two separate training set concepts and accordingly, two separate training subsets can be created, denoted as the model training set and the recommendation training set. These two subsets differ with respect to the point in time at which they are provided to the computing environment. The model training set is provided to the algorithm when the computing environment is started in order to bootstrap the recommendation engine (e.g., the ratings to compute the Singular Value Decomposition of a collaborative filtering algorithm). On the other hand, the recommendation training set is composed of all data provided to the recommendation algorithm only when a recommendation is requested. Note that some of the three subsets can be empty (e.g., the recommendation training set might be missing) and, in addition, some subsets can be overlapped.

Evaluator capabilities

Splitting the dataset into the aforementioned three parts makes it possible to cover most existing evaluation strategies. Note that the three parts can also be overlapping (e.g., the information about a user used for optimizing the algorithm might also be sent to the engine when the recommendation is requested for such a user), or some subsets might be empty.

As an illustrative example, let us assume that the evaluation strategy requires to separate users used to optimize the algorithm from users used to compute the evaluation metric. For

instance, 50% of users is used to tune the algorithm, and the remaining 50% will be tested (but are not to be used to optimize the algorithm) according to an “all-but-5” protocol, i.e., each user is hidden 5 items. The dataset will be split as follows. All the ratings given by 50% of the users will form the model training set. Therefore, for each one of the remaining 50% of users, 5 ratings will be hidden and will form the test set, the remaining ratings will be included in the recommendation training set.

As already identified in D2.1 “*First Iteration Requirements plus Evaluation Specifications*” (Section 8.3, page 44), the two main goals of the recommendation training set are (i) to evaluate stream recommendations and (ii) to recommend entities on-the-fly.

4.3. Computing environment

The computing environment (typically, a virtual machine) contains the environment to execute an algorithm and the implementation itself. As an example, it contains the recommendation algorithm and all the required libraries. Among the other functionalities, the computing environment must be able to serve recommendation requests and to provide some system statistics (e.g., cpu times, i/o activity).

The output format of the computing environment is specified in Section 5.2.

4.4. Orchestrator

The orchestrator is in charge of initiating the virtual machine, providing the training data (both model and recommendation) at the right time, requesting the recommendations, and eventually collecting the results to compute the evaluation metrics.

The communication protocol between the orchestrator and the computing environment is specified in Section 5.3.

4.4.1. File-based Orchestrator

The first proof-of-concept of the orchestrator was fully based on file-based messages. In the practice, sending a message means that the sender writes a file in shared directory (the file, for instance a textual file, contains the body of the message). On the other side, the receiver of the message has to regularly check the shared directory for the presence of a new file and, when found, reads the file and the related message. Once the message has been read, it can be removed from the shared directory.

The body of the message might contain some pointers to other data (e.g., the training data set). Essentially, it means the the body refers to an other file (stored in the shared directory) that the receiver has to access in order to retrieve the data. This decouples the message (i.e., the instruction/command to execute) from the data.

The file-based solution to exchange messages is quite simple to be implemented, however it has some drawbacks, the main one being the fact that the sender has to constantly pool the shared directory, resulting in an intrinsic delay between when the message is sent and when it is received, thus preventing from being used in real-time scenarios. Furthermore, the file-

based communication requires a certain effort to manage the communication of multiple, concurrent messages.

4.4.2. Message-based Orchestrator

Although the previously described file-based Orchestrator solves the task of controlling the various parts of the reference framework in a charmingly simple way, as mentioned above one of its drawbacks is the introduced delay between a command and the actual execution of it due to the polling nature of the architecture.

Hence, we have defined the following requirements for the messaging system of the Orchestrator:

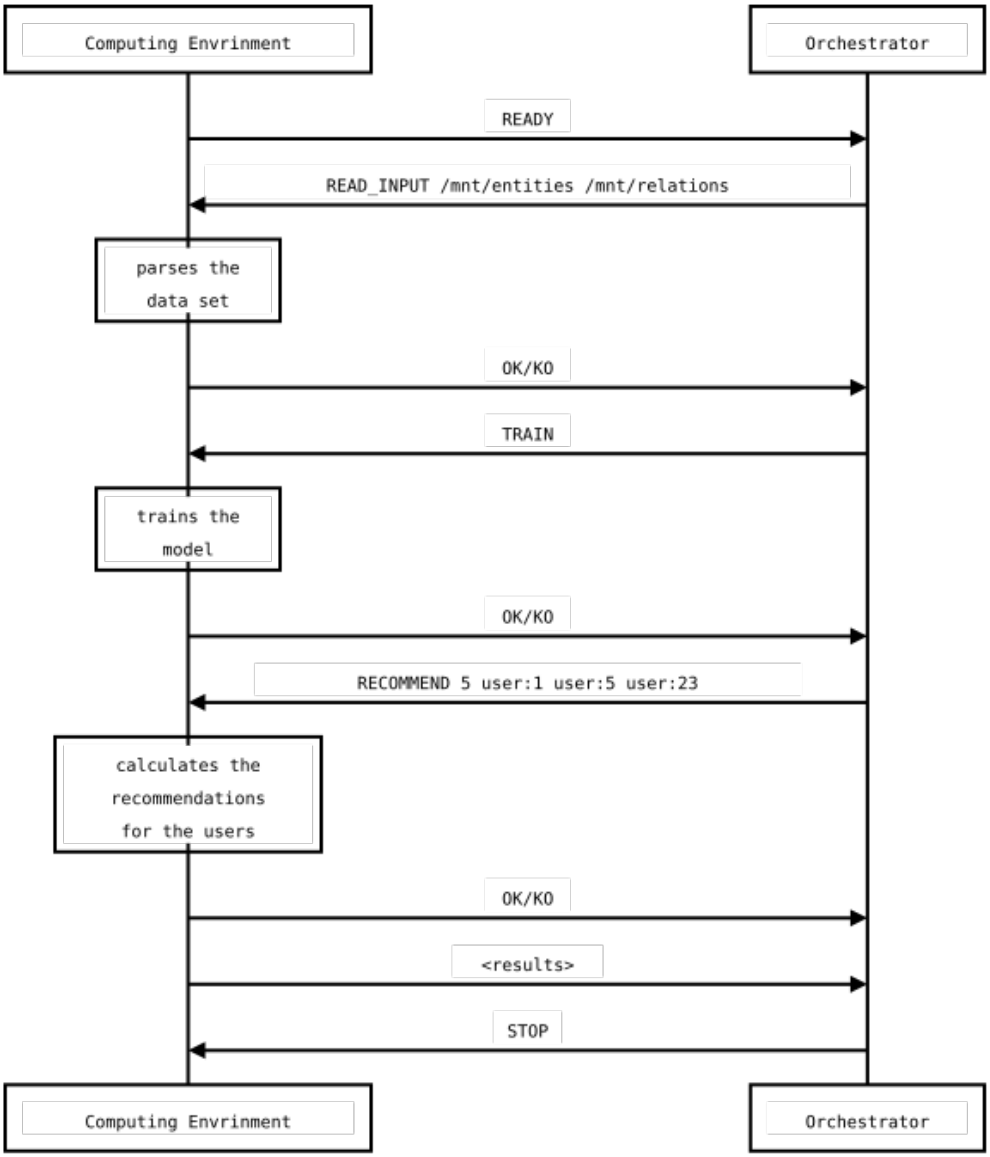
- (network) socket based
- push strategy; the only delay between a command and its execution is due the inevitable networking lag.
- simple and portable

Instead of defining a low-level networking protocol using sockets that later should be implemented in each component of the framework from scratch using various libraries available, we chose to use a very tiny networking library called 0MQ, which

- supports every modern language and platform
- carries messages across **inproc**, **IPC**, **TCP**, **TPIC**, **multicast**
- high-speed asynchronous I/O engines

0MQ enables the developers to rapidly implement the protocol of the reference framework in a custom algorithm, as regardless of the actually programming language being used, the API calls of 0MQ are almost identical.

The 0MQ based orchestrator implements the communication protocol described in Section 5.3. with a client-server architecture using **TCP** protocol. Figure below shows the whole sequence of messages between the Orchestrator and the Computing Environment, that basically trains and evaluates a recommendation algorithm using a benchmark data set.



Once the Orchestrator has been started it initiates the starting of the computing environment and in the meanwhile starts listening to a specific **TCP** port. As soon as the Computing Environment is ready for processing it should notify the Orchestrator by connecting to that **TCP** port and send a **READY** message to the Orchestrator. Once the **READY** message received by the Orchestrator, it sends the location of the benchmark data set to the Computing Environment with the **READ_INPUT** message. For the details of the data set format see the next section. Once the data set was successfully read by the algorithm the Orchestrator first issues a **TRAIN** command to train the model and then requests recommendations for a given set of user with the **RECOMMEND** command. Finally the Orchestrator sends a **STOP** message to the Computing environment to shut it down.

As one can see the data set itself is not transferred via the 0MQ channel, but rather stored in a shared directory (shared between the Orchestrator and the Computing Environment) and only the path and name of the (shared) files are exchanged. We have been experimenting to use the 0MQ channel to distribute the data set itself as well. FileMQ, a publish-subscribe file service based on 0MQ, seemed like a good choice for transferring data sets via 0MQ channel, but:

- it is not as portable as 0MQ, i.e., as of now FileMQ supports only a handful of programming languages
- seeking is not supported; In case of a file shared via a folder one does not need to transfer the whole file in order to be able to seek within the file itself. Whereas in case of FileMQ or any other similar solution, where the file is transferred as part of the read command, one needs to read and transfer the whole file prior seeking within the file would be possible.

Due to these limitations we decided to distribute the data set via shared folders instead of transferring it via 0MQ. Although it is worth to note that one of the limitation of using shared folders and files to distribute data sets is the lack of support for streaming input.

5. Specifications

5.1. Data model

A dataset is represented by means of a *multigraph*, i.e., a set of **nodes** and **links**. In a multigraph, a link connects together two nodes, and two nodes might be connected by multiple links. The multigraph models:

- **entities** that correspond to the nodes of the graph. The main entities are users, items (e.g., a song, a movie), and contexts (e.g., location, device,..). Each entity is described by a set of properties.
- **relations** that correspond to the links of the graph. A relation connects two entities together, and has a set of properties, such as when the relation started and finished (in the case of non-instantaneous actions) and the contexts the relation happened (e.g., from the user's smartphone). Example of relations are: the user rated an item 4 out of 5, the user watched a movie for 80% of its play time, or a user became a friend of another (in the case of social network connections).

Entities and relations are modeled as follows.

5.1.1. Entities

An entity is a piece of information that can be modeled and recommended. An entity is represented by: a type (e.g., movie, user, person, genre), an identifier, a set of properties (e.g., the title of a movie), and a set of links to other entities (e.g., the actor of the movie is represented with a link to the 'person' entity "Kate Winslet").

Note that, if we want to recommend movies according to conventional, pre-assigned genres, then the genre of a movie should be an entity, and the movie–genre assignment is modeled by a relation.
On the other hand, the movie title can be represented by a simple property (being not object of a recommendation).

Entities can be stored in textual files according to the following format:
etype <TAB> **eid** <TAB> **timestamp** <TAB> **properties** <TAB> **linked_entities**
where:

- **etype** is a *String* representing the type of entity (e.g., movie, book, person)
- **eid** is a *String* representing the id of the entity
- **creation_time** is a *Long number* the Unix epoch time expressed in milliseconds (UTC) The creation_time can be empty (e.g., in the case of static dataset)
- **properties** is a *map* of key-value entries, formatted as a JSON map, where
    - key is a String
    - value is either a String, a Number, or an array of Strings/Numbers
- **linked_entities** is a *map* of key-value entries, formatted as JSON map, where
    - key is a String
    - value is a linked entity, represented by a String in the form **"etype:eid"**

In addition:

- the **TAB** character cannot be used
- the **CR and/or LF** characters cannot be used
- the **etype** and **eid** cannot contain **colons**

JSON is to be formatted according to standard format, using *double quote for strings*

5.1.2. Relations

A **relation** is defined by: a **type** (e.g., view, rating), an **identifier**, a **timestamp**, a set of **properties** (e.g., the score of a rating, the playtime), and a set of **links to entities** (e.g., the user who expressed the rating, the related item, the device, etc.). A relation represents a connection that links multiple entities (e.g., the rating given by a *user* to a *movie* at *home* using a *smartphone*), differing from a linked entity (dashed line) that represents a fact (e.g., the *movie* has an *actor*).

5.1.3. Examples

The data model proposed in CrowdRec enables to encode the all typical user feedback types, such as the four canonical scenarios depicted in Figure 1:

- **explicit feedbacks**: user1 explicitly rates a movie1 with a rating of 3 stars out of 5
- **implicit feedbacks**: user1 plays track1 for 97 seconds

- **social relationships**: user2 has 2 friends, namely user1 user3 and he/she is followed by user4.
- **context data**: user1 watches movie1 on his TV in Milan.

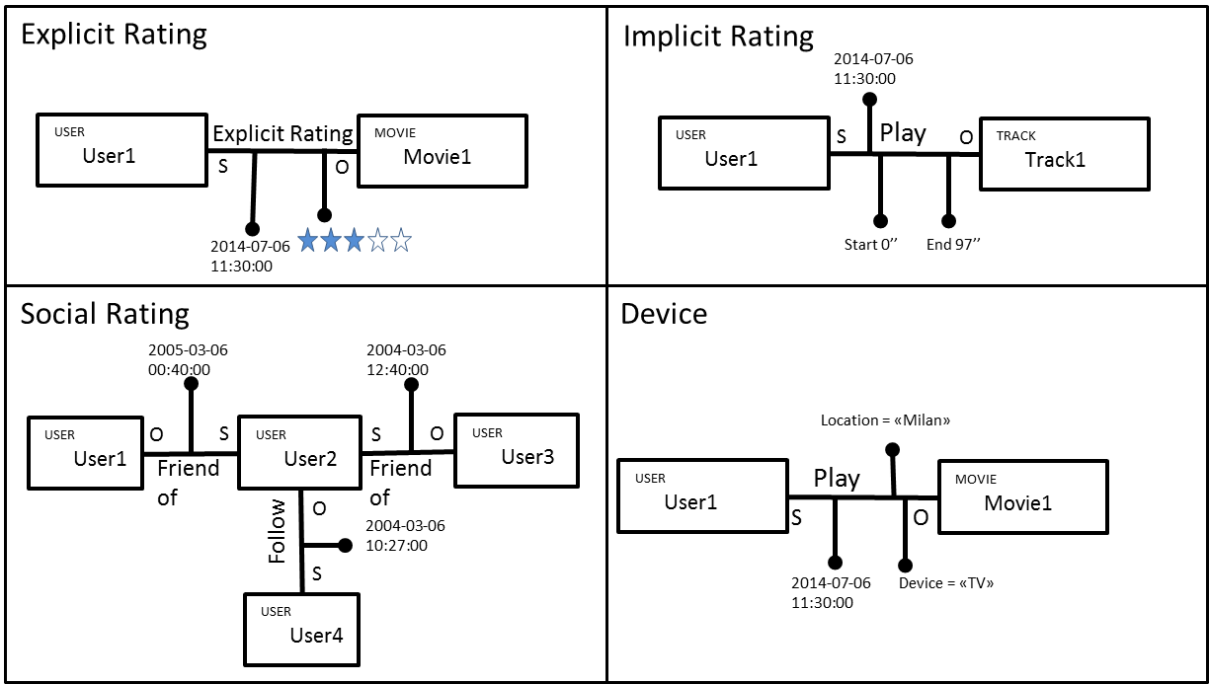


Figure 1 - Data model: use cases

5.2. Output format

The output of the recommender algorithms is structured in a similar format as the input file. Output files are stored in a TSV format. The tab-separated columns can represent data as JSON. Columns not required for the evaluation can be kept empty. The minimal information, need in each line of the output file is an unique id referring to the request and computed prediction (usually represented in JSON format). Additionally, the output might contain an explaination, allowing the user to understand, why particular items have been recommended.

Using a similar data format for the input and the output files allows us using the same parsers for the input and output files. In addition, it also simplifies the analysis of results based on N-fold cross-validation.

5.3. Communication protocol

This section describes the communication protocol between the orchestrator and the computing environment, divided on the basis of the operation to perform. Each communication consists of a source component (sender), a destination component (receiver), and a message. The message might depends on some parameters. Most messages require the receiver to reply to the sender. Only the “Computing Environment booting” message and the “Computing Environment shutdown” message do not require a reply.

Computing Environment booting

The computing environment notifies the orchestrator once it completes booting.

Sender: computing environment

Receiver: orchestrator

Message: READY

Parameters: -

Computing Environment shutdown

The orchestrator orders the computing environment to shutdown.

Sender: orchestrator

Receiver: computing environment

Message: STOP

Parameters: -

Read input data

The orchestrator notifies the computing environment that input data (entities and relations) are available at a certain path on the shared folder.

Sender: orchestrator

Receiver: computing environment

Message: READ_INPUT <path_entities> <path_relations>

Parameters:

- <path_entities> is the path of the file containing the entities
- <path_relations> is the path of the file containing the relations

The computing environment replies to the orchestrator with a *status message*, either OK (success) or KO (failure).

Sender: computing environment

Receiver: orchestrator

Message: OK (or KO)

Parameters: -

Train recommender algorithm

The orchestrator orders the computing environment to train the implemented recommendation algorithm (on the basis of the data received so far)

Sender: orchestrator

Receiver: computing environment

Message: TRAIN

Parameters: -

The computing environment replies to the orchestrator with a *status message*, either OK (success) or KO (failure).

Sender: computing environment

Receiver: orchestrator

Message: OK (or KO)

Parameters: -

Request for recommendation

The orchestrator requests the recommendations for a list of users to the computing environment.

Sender: orchestrator

Receiver: computing environment

Message: RECOMMEND <#recomLength> <entity_1> <entity_2> ... <entity_M>

Parameters:

- <#recomLength> is the number of entities to be recommended
- <entity_i> is the identifier of the i-th entity to be recommended

The computing environment replies to the orchestrator with a *status message*, either OK (success) or KO (failure), followed by the one *recommendation message* for each entity for which the recommendation was asked for. The *recommendation message* consists of the list of recommended entities (represented by their identifiers).

Sender: computing environment

Receiver: orchestrator

Message: <recomm_entity_1> <recomm_entity_2> … <recomm_entity_N>

Parameters: <recomm_entity_i> is the identifier of the i-th recommended entity.

6. Automatic provisioning

The provisioning of virtual machines is managed by Vagrant that allow to create a reproducible, consistent and portable (OSX, Linux and Windows) environment for the execution of the algorithms. Furthermore Puppet is used to manage operative system configuration and software prerequisites installation. The combination of Vagrant and Puppet let the reference framework execute the full orchestration starting from a set of configuration files that are stored in the reference configuration management software (GitHub) and guarantees the consistency of the full execution software stack.

Vagrant

Vagrant controls the full workflow for the provisioning of the virtual machine and makes possible the configuration of a single or a set of machines that can run on VirtualBox, Vmware or Amazon web services.

The current reference framework standard execution environment is based on VirtualBox. This does not, however, limit in any way the possibility of configuration and execution on different providers, allowing users to configure and compare algorithms even on different physical environments.

Moreover Vagrant machine configuration allow the reference framework end user to execute their recommendation algorithms on different environment scaling that both in a vertical way (configuration of memory/CPU) and horizontally (adding new nodes) and enabling in such a way also the evaluation of algorithms performance.

Vagrant open source repository is present on GitHub at the following URL: https://github.com/mitchellh/vagrant

Puppet

Puppet is one of the standard provisioning software that can be used out-of-the-box within Vagrant. With the objective of standardize the deliverables of the project, all the computing environments that will be developed within CrowdRec will use Puppet as standard provisioning software.

The choice of Puppet is supported by the following:

- has a large base of installations
- large developers base
- a comprehensive repository for configuration modules
- support for os X and linux
- Apache 2.0 license

The first release of the reference framework provided a working vagrant and puppet configuration for the deployment of a computing environment based on the Mahout framework.

7. Reference framework release

7.1. Computing Environments releases

The reference framework is designed to be compatible with most existing frameworks, both public, open frameworks (e.g., Mahout, MyMediaLite, etc.) and custom, proprietary frameworks.

Some common computing environments will be released together with the reference framework in order to facilitate the spread of the CrowdRec's reference framework by allowing to immediately experiment with algorithms already implemented in one of these frameworks. The choice will be driven by both the requirements developed in Task 2.1 (c.f. D2.1 "*First Iteration Requirements plus Evaluation Specifications*") and the algorithms developed in WP3 "*Stream Recommendation Algorithms"* and WP4 *"Crowd Engagement Algorithms"*.

The first release of the reference framework includes the configuration for the recommendation framework WrapRec. WrapRec (https://github.com/babakx/WrapRec) is an easy-to-use Recommender Systems toolkit which allows users to easily implement or wrap recommendation algorithms from other frameworks. WrapRec is implemented in C# and is publicly available under GPL license in github. The algorithms that are developed in WrapRec can be used either directly or through CrowdRec reference framework. WrapRec can be directly use through installing its Nuget package or by importing it as a dll library in any .Net-based solution. Installation of WrapRec nuget package is decribed here: http://www.nuget.org/packages/WrapRec.dll/

Furthermore, for the future we plan to include the configurations of additional existing frameworks; most solutions described and referenced in Appendix A (e.g., Okapi) can be easily integrated should it be necessary.

7.2. Evaluator release

The default evaluator of the reference is RiVal (http://rival.recommenders.net) which is integrated into the reference framework as a Maven dependency. The reference framework instantiates the necessary evaluation metrics and strategies from RiVal and executes the compiled code. RiVal currently interfaces with the reference framework through files located in folder which are shared between the computing environment and RiVal, i.e., RiVal creates the training and evaluation files which are used by the recommendation frameworks in the computing environment. Recommendation are written into files, and these are subsequently fed back to RiVal again in order to perform an evaluation (see Figure 2 - The RiVal pipeline. The reference framework interacts with RiVal between the Split and Recommend phases, and then again between the Recommend and Candidate Items phases.).

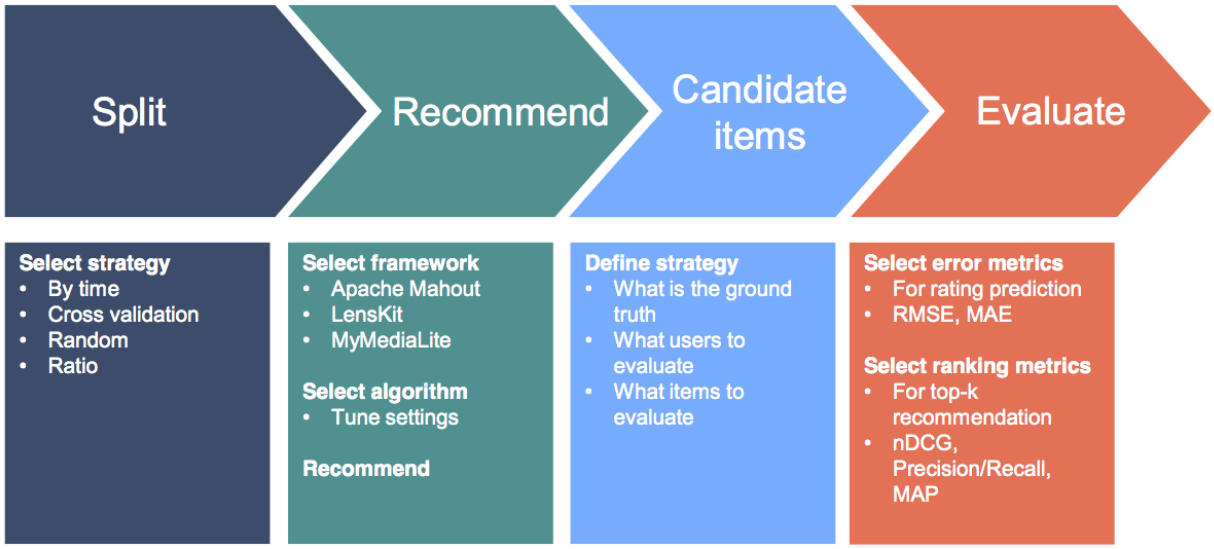


Figure 2 - The RiVal pipeline. The reference framework interacts with RiVal between the Split and Recommend phases, and then again between the Recommend an Candidate Items phases.

The evaluation step is implemented as a minimal program in the reference framework, this program specifies which evaluation metrics should be used, and at what levels of recall. RiVal currently includes standard metrics used in recommender systems and information retrieval.

The RiVal has a separate release line from the reference framework to allow for more dynamic implementation, e.g., additional evaluation metrics, etc.

7.3. Data release

The first release of the reference framework is accompanied by a version of the MovieTweetings dataset (https://github.com/sidooms/MovieTweetings) and converted into the data format adopted by the reference framework (the dataset is available in github at https://github.com/crowdrec/datasets/tree/master/01.MovieTweetings). See Section 2 for further details about the dataset.

8. Reference framework dissemination

We created a dissemination web site for the reference framework release: http://rf.crowdrec.eu/.

The web site primary aim will be to describe the framework's objectives, create a community of users and explain how it can be used and integrated.

The development and release of the reference framework is supported by the collaborative platform GitHub. We created the organization "CrowdRec" (https://github.com/crowdrec/) which is composed by 5 repositories, one for each macro software component of the reference framework architecture (see Reference framework architecture):

- the orchestrator that manages the whole reference framework: https://github.com/crowdrec/reference-framework
- the available datasets in the CrowdRec data format: https://github.com/crowdrec/datasets
- the released computing environments: https://github.com/crowdrec/computingenvironments
- the implemented algorithms running within one (or more) computing environments: https://github.com/crowdrec/algorithms
- the evaluators that can be used to experiments with the released algorithms: https://github.com/crowdrec/evaluators

The reference framework has been presented at the IBC conference and a paper work describing it has been included in the conference proceedings [3]. Furthermore, CrowdRec attended NEM (New European Media) initiative (Brussels, 29-30 September) presenting the first release of the reference framework to the public. More information about IBC and NEM are available in D6.2 "First year dissemination report" (Section 2.4, page 8).

9. Reference framework evaluation

We evaluated the framework from three aspects. First, we investigated how the current version satisfies the requirements set in the conceptual design phase. Second, we organized a meeting held in Milan where the first version of the reference framework was tested by the participant with the help and support of the developers, and was consequently evaluated from the usability and technical integrity point of views. Third, a conceptual business evaluation was performed by recommender system vendors regarding the business applicability of the framework in the various segments of the potential business users, such as TV operators and players in the online industry.

9.1. Compliance with the requirements

In the Requirements Section 3.1 we identified three major requirements the reference framework should satisfy:

- **Consistency and reproducibility**
- **Architecture independent**:
- **Standardization**

On *consistency and reproducibility* we meant that the reference framework should implement a consistent environment to run and evaluate quality, performance and scalability of algorithms. Also, the reproducibility of experiments and related results should be guaranteed.

This requirement dictated the major design concept of the reference framework. As a consequence, the reference framework decouples the various roles of the recommendation task, such as

- data set provider;
- computation environment that is responsible to provide the recommendations, also, in the most typical model based approaches this component builds and maintains the recommendation model;
- evaluation of recommendations;
- orchestrator that connects the independent components, and enables to provide an end-to-end solution for the recommendation tasks.

Evaluation of the capability to handle various recommendation tasks

The main component of the reference framework is the orchestrator. We experimented with two messaging technologies to perform the communication between the orchestrator and other components. First, we evaluated the file based orchestrator, which is extremely simple to implement, but it has the drawback that due to its polling mechanism there can be a significant lag between the message sending and receiving, which hinders its application for real-time recommendation tasks. Therefore, we opted for the messaging based orchestrator that enables real-time recommendation task and the only lag between a recommendation call and its receipt is caused by the inevitable network latency.

Therefore we state that the messaging based *orchestrator satisfies the requirement that the reference framework should be able to perform different types of recommendation tasks* (notably: online and offline).

Evaluation of quality and performance

The quality and performance evaluation of the algorithms are executed by the Evaluator (EVL) component. By decoupling the algorithmic part (computing environment) and the evaluation guarantees that

1. the evaluation is consistent if the same evaluator component is used
2. independence from the implementation of algorithms
3. reproducibility of the experiments provided that the same dataset is used.

Worth recalling that to guarantee the consistency of the evaluation it is necessary to select a single default evaluator toolkit.

We consider the advantage of the conceptual design of the reference framework that the Evaluator can be replaced or several Evaluator can be instantiated. This equally enables to switch to an alternative new Evaluator to be developed by the open source community in the future, and at the same time evaluation of the Evaluators along the standard metrics used in recommendation tasks.

For the first release of the reference framework, CrowdRec opted for the RiVal toolkit as the default evaluator. This is actually a choice that can be criticized since the functionality and the technology of the evaluator restricts the potential capabilities and determines certain technological aspects.

When selecting RiVal, we therefore considered that it is written in Java, thus being platform independent, implements all the major evaluation scenarios typical for recommendation task, is easy to extend, and it is a living open source project with the necessary support within and beyond the CrowdRec community.

One trivial restriction of RiVal is that it reads the entire dataset into memory for certain tasks (e.g., dataset splitting), which by design limits its applicability for huge dataset not fitting into the memory. Although this is a clear drawback, to our best knowledge, the alternative evaluation toolkits also suffer from the same problem, and the existing support of RiVal in the consortium enables to waive this restriction in the next release of RiVal and the Reference framework itself.

Evaluation of the algorithm scalability

The algorithms' scalability is designed to be evaluated by two components. First, the Orchestrator measures the elapsed time of tasks performed by the computational environments. Note that the messaging based orchestrator enables the validity of such measurements, since only the network latency is added as extra. Second, the computing environment is expected to provide internal system statistics (CPU time, I/O activity) on the algorithmic part. The most important scalability characteristics are summarized in the table below.

Recommendation task	Data size	Response time
model building, offline recommendation	• Determined by ORC • Check the ability to cope with huge data sets to train • Critical for data scalability and to use at industry scale	• Measured by CPE and ORC • Check the ability to complete recommendation task within reasonable time • Not critical (more flexible requirements)
real-time recommendation	• Determined by ORC • Check the ability to cope with large traffic • Critical to scale to industry standards	• Measured by ORC • Critical to scale to industry standards

Note that in the first release of the reference framework the real-time recommendation measurement is not supported yet, it will be covered when stream recommendation is implemented.

We state that the *conceptual design of the reference framework enables to measure the ability to fulfill the scalability requirements of algorithms implemented in the CPEs*.

Evaluation of the support of reproducibility

The reference framework architecture supports the reproducibility of the experimentation through the standardization of the implementation and the evaluation, including experimentation setting. By the standardization of the implementation we mean that the popular recommendation frameworks and toolkits will be integrated with the reference framework therefore the same implementation of basic algorithms will be available to the public. On the other hand, the Evaluator guarantees that the same experimental setting can be activated, which then supports the reproduciblity of the entire experimentation provided that the dataset is available.

Evaluation of architectural independence

The architectural independence is a crucial requirement that is necessary for the uptake of the reference framework in the research and open source community and in the industry. The reference framework guarantees architectural independence because the different components are decoupled and the communication between the components is performed via standardized interfaces, namely the data format layer and the messaging mechanism.

The data format layer is a generic multigraph model that represents the data by entities and their relations. The reference framework provide a JSON API through which such formatted data is made available to algorithms. To be able to process the data, wrappers should be implemented in the computing environments. For the most popular recommendation toolkits, the CrowdRec consortium will implement the wrappers for demonstration and example.

As for messaging mechanism a very tiny networking library, 0MQ, was selected that supports every modern language and platform. This also guarantees that all consortium members (and further practitioners) are able to implement their algorithms in their preferred architecture/language. The fact that the API calls of 0MQ are almost identical, also significantly facilitates the rapid implement of the protocol to the reference framework, in addition to the reference implementations to be provided to most popular toolkits.

In addition to the communication layers, the platform independency is supported by the virtual machine technology we apply for managing computing environments. Vagrant that manages the provisioning of virtual machines allows to create a reproducible, consistent and portable (OSX, Linux and Windows) environment for the execution of the algorithms. Puppet is used to manage operative system configuration and software prerequisites installation.

We thus state that the conceptual design of the reference framework efficiently supports architectural, language and platform independence.

Evaluation of data standardization

Comparison of experiments necessitates to run algorithms on the same data, be it offline or online evaluation.

The data standardization requirement is satisfied by two factors of the architectural design. First, the data model guarantees that different data sources are represented in a unified format that is accessible to algorithms via a standard JSON API. Second, the Evaluator enables to pre-process raw data in a configurable but fixed manner and then feed the each algorithm with the same data chunk.

For online evaluation, the same standardization is concept is needed to be implemented for stream data in the next release of the reference framework in order to guarantee full compliance with the standardization requirement.

We state that *the current release of the reference framework satisfies the offline data standardization concept, and the same concept should be carried over to the next release for handling stream data*.

9.2. Internal evaluation of the reference framework

The first version (v0.1) of the Reference Framework was released internally in May 2014. We performed the first internal evaluation during the meeting held in Milan at Moviri headquarter early June 2014. The main goal of the meeting was to conduct a hands-on session with the developers of the reference framework for the technical staff of consortium members who participate in this work package.

We held a session dedicated to the implementation of the various components of the reference framework and the supporting software (Vagrant, Puppet). Working on different platforms (Linux, iOS, Windows), participants could directly evaluate the usability of the system and could report difficulties to developers.

We identified for instance that certain scripts provided by the organizers do not work on Windows or need additional utilities to be installed for proper functioning. Such critiques helped the improvement of the implementation of the first release.
We also had discussion on the conceptual design of the framework. As a result of this, we identified the drawback of the file based orchestrator, and we formed a working group to substitute that with messaging based orchestrator, being released in the current version.
We also plan similar to organize similar activities in the future for the internal technical and conceptual evaluation of the reference framework, as well as to integrate the feedback of the EAB and the community.

9.3. Evaluation of business applicability

The industry participants of CrowdRec, including recommender vendors, Moviri and Gravity, are planning to use the reference framework for evaluating their in-house implementations in comparison with open source software, and also to provide support for their client in conducting such evaluations. Therefore it is crucial to satisfy the requirements such external evaluators may require. We therefore evaluated the reference framework from the point of view of a potential industry user.

- **Data privacy:** industrial partners will use the framework if their data will not be distributed to 3rd party without their consent.
- **Ability to compare the in-house solution:** companies may have their in-house solution for recommendation that they want to compare with others without providing access to their own system
- **Ability to test 3rd party vendors:** aligned with the above requirement they may want to test 3rd party vendors by providing them test data dedicated for such purpose.

The reference framework supports data privacy of industry partners. For internal use the reference framework can be installed within own data center data and in-house and open-source solutions can be there readily tested. No data needs to leave the data center. For testing 3rd party vendors the data can be made available to the participating vendors by password protected link thus it is guaranteed that the only the selected vendors can access the data. If the orchestrator is installed in such a case at the testing partner, also the scalability aspects of algorithms can be directly tested.

10.Conclusions and outlook

This deliverable has presented the first release of the reference framework, together with the first evaluation process. It has also discussed related datasets and independent implementations of CrowdRec algorithms accompanying scientific publications in the algorithms workpackages (i.e., WP3 and WP4).

The requirements collected in D2.1 "*First Iteration Requirements plus Evaluation Specifications*" have been translated in practical properties the reference framework has to

fulfill. The first release of the reference framework implements all the components that have been designed in order to have an effective environment to (i) grant consistency and reproducibility of results, (ii) architecture independence, and (iii) standardization.

The first release can be considered *proof-of-concept* of the Reference Framework (named 'idomaar'). It as it has been released with a data set illustrating the CrowdRec data model, a limited set of algorithms (among them, one has been selected from WP3), and a simple evaluator strategy, all coordinated by a basic orchestrator.

Each component (e.g., the dataset, the evaluator, etc.) has been designed so to be enough generic/abstract to allow for custom implementations. Consequently, next iterations will expand each module to enrich the set of available options. Specifically:

- We will enlarge the available set of datasets. One possible candidate is the SoundCloud comments dataset, already analyzed in Section 2 and goes into the direction of evaluating the use case scenarios concerning the media domain described in *WP5 "Large-scale real-world application and deployment".*
- We will extend the set of *"pre-configured"* computing environments, so that practitioners relying on standard frameworks (e.g., Mahout) can easily experiment with their implemented algorithms using one of the existing environments. This activity will consequently extend the set of available algorithms to all the ones included in the existing framework.
- We will implement additional algorithms that have been already implemented or will be designed during next iteration in *WP3 "Stream Recommendation Algorithms".*
- We will add additional evaluation strategies and metrics.
- We will add the full support for streams of data. The capacity to manage streams of data has already been designed (for example, the choice of using ZeroMQ for the communication between the orchestrator and the computing environment). Next releases will implement and test it.

Further priorities for the reference framework will be defined on the basis of the requirements that are established in Task 2.1.

APPENDIX

A. Related initiatives

We identified four main types of solutions, according to their purpose: (i) simple generic libraries, (ii) recommendation libraries, (iii) generic machine learning frameworks, and (iv) frameworks specifically focused on recommender systems.

A.1. Generic libraries

Generic libraries provide methods for the efficient implementation of algorithms and the efficient handling of data. In contrast to complete frameworks, these libraries can be adapted to fit the specific requirements of a specific algorithm. Generic libraries, we will use in the project are Guava and SVDfeature.

Guava (http://code.google.com/p/guava-libraries/), a successful library written in Java (as of April 2012, ranked among the top 12 most popular Java libraries) that provides data structure to facilitate the development.

SVDfeature (http://svdfeature.apexlab.org) is a C++ library to support SVD-based algorithms.

A.2. Recommendation libraries

Crab (https://github.com/muricoca/crab) and **Python-recsys** (https://github.com/ocelma/python-recsys) are two Python-based recommendation libraries to support the development of recommender systems.

Recommendable (https://github.com/davidcelis/recommendable) is a gem (i.e., a library) for Ruby applications that provides some trivial recommendation services.

A.3. Generic machine learning frameworks

Mahout (https://mahout.apache.org/), a well-known and active Java-based Apache project that provides the implementations of a set of distributed and scalable machine learning algorithms. A few recommendation algorithms are already implemented (e.g., user-based, item-based, svd-based collaborative filtering). Scalability is grant by the map/reduce paradigm. In addition, Mahout provides support to the evaluation of algorithms thanks to interfaces to split dataset and compute accuracy/error metrics.

GraphLab (http://graphlab.org/) is a C++ based graph-based data mining tool oriented to highly-scalable and sparse-data problems. GraphLab was created (2009) to develop a new parallel computation abstraction tailored to machine learning. It has a large selection of ML

methods already implemented (http://docs.graphlab.org/toolkits.html) and you can implement your own algorithms on top of GraphLab API (C++).

Okapi (https://github.com/grafos-ml/okapi) is a Java-based solution that provides machine learning and graph mining algorithms for Giraph. Okapi is part of the grapfos-ml project (http://grafos.ml/), whose objective is the developmentt of tools for large-scale machine learning and graph analysis. Okapi is available under the Apache2 licence.

Gravity Data Mining Framework is a proprietary Java-based data mining framework developed by one the CrowdRec's partner - namely Gravity - that is used to experiment with the off-line evaluation of recommendation algorithms, the development and optimization of new recommendation approaches, and proof-of-concepts. A few algorithms have already been implemented, e.g., matrix and tensor factorization, content-based filtering, popularity-based method, etc. The access to the framework can be provided upon request in the consortium based on the consortium agreement.

A.4. Recommendation-oriented frameworks

EasyRec (http://easyrec.org/) is a Java-based recommendation framework that exposes its services as SaaS. It is mainly designed for simple integration into existing web site. It implements a few recommendation algorithms (e.g., also viewed/bought, user recommendations, related items, item clustering, top viewed).

Mrec (https://github.com/mendeley/mrec) is a Python-based recommendation framework that offers some implemented state-of-the-art algorithms together with testing capabilities.

RecDB (https://github.com/Sarwat/recdb-postgresql) is a recommendation solution based on the database *postgresSQL* implementing some collaborative filtering algorithms.

MyMediaLite (http://www.mymedialite.net/) is a multi-purpose framework implementing multiple recommender system algorithms. It is implemented in .Net (C#). It runs on every architecture supported by Mono (Windows, Linux, Mac OS X), using the dll library. The framework addresses most scenarios in collaborative filtering, such as rating prediction and item rank prediction. It also provides evaluation mechanisms (e.g., to compute MAE, RMSE, AUC,...).

Myrrix (https://github.com/myrrix/myrrix-recommender) was an interesting project build on Apache Mahout to provide real-time and scalable recommender systems. The project has been discontinued since 31st December 2013.

LensKit (http://lenskit.grouplens.org/) is Java-based framework that provides both a set of implemented recommendation algorithms and a number of evaluation strategies. It does not address scalability, so it mainly targets small-medium size datasets.

Test.fm (https://github.com/grafos-ml/test.fm) is a testing framework for collaborative filtering. As well as Okapi (see Appendix A.3), also Test.fm is part of the grafos-ml project (http://grafos.ml/).

RiVal (http://rival.recommenders.net/) is Java-based toolkit for recommender system evaluation which allows the comparison of results across different recommendation frameworks.

WrapRec (https://github.com/babakx/WrapRec) is an easy-to-use Recommender Systems toolkit, written in C#, which allows users to easily implement or wrap recommendation algorithms from other frameworks. WrapRec provides a rich data model which makes it easy to implement algorithms for different recommender system problems, such as context-aware and cross-domain recommendation.

B. References

[1] Simon Dooms, Toon De Pessemier, Luc Martens. “MovieTweetings: a Movie Rating Dataset Collected From Twitter” in *Proceedings of Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013.*

[2] Alan Said, Simon Dooms, Babak Loni, and Domonkos Tikk. “Recommender Systems Challenge 2014” in *Proceedings of the eighth ACM conference on Recommender systems (RecSys 2014)*. Foster City, CA, USA, ACM 2014

[3] D. Tikk, R. Turrin, M. Larson, D. Zibriczky, D. Malagoli, A. Said, A. Lommatzsch, V. Gál, and S. Székely, “Comparative Evaluation of Recommendation Systems for Digital Media,” in *Proceedings of the IBC conference 2014*, Amsterdam, Netherlands, 2014.