Knowledge-Based Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Original software publication

RecoLibry-core: A component-based framework for building recommender systems

Jose L. Jorro-Aragoneses [*], Juan A. Recio-García, Belén Díaz-Agudo, Guillermo Jimenez-Díaz

Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Madrid, Spain

ARTICLE INFO

Article history:
Received 13 March 2019
Received in revised form 5 June 2019
Accepted 16 July 2019
Available online xxxx

Keywords:
Recommender systems
Component-based development
Java

ABSTRACT

Recommendation systems are a key part of almost every modern consumer website. These systems include techniques to filter, explore and rank a huge amount of information based on users' preferences or similar items. Designing and implementing a recommender system from scratch require skills of programming and recommending technologies. In this paper we describe RecoLibry-core, a framework to develop recommender systems based on the reuse of components provided by third-party frameworks.

1. Introduction

Recommender systems (RS) represent a very successful family of systems that explore and filter knowledge about items and users to predict the preference that a certain user would give to an item. Recommendation is based on a number of techniques that have been proposed in the literature (see [1] for a comprehensive review) and have been applied to many different domains of applications like movies, books or music [2].

The design process of a RS from scratch is a complex task where many decisions are taken. The system designer should be able to choose the most appropriate recommendation algorithm and configuration parameters. An expert designer would require an in-depth analysis on the available data and the behaviour of the algorithms. Besides, the choice depends on many different factors, such as the type of knowledge about the items and the target users, the data structure, the existence of social or contextual knowledge, the performance and the size of the knowledge base, and others.

In recent years, numerous frameworks have been created to make RS [3–6]. However, in most of these frameworks there are two major problems. First, many frameworks are oriented to a single type of recommendation methods such as Lenskit [3] (focused on collaborative filtering algorithms) or Tensorrec [6] (focused on machine learning). The second problem of these frameworks is that they are oriented to users with previous knowledge in the development of such type of systems. They do not offer non-expert users the guidance required to design or deploy recommender systems. In addition, an active research area is made tools to integrate different algorithms, for example [7], or tools to explain users algorithms or data used in a system [8].

In this paper we present a tool to solve these problems. RecoLibry-core is a Java framework to create RS by reusing components. It is a tool included in RecoLibry Suite, a set of intelligent tools to build RS. RecoLibry-core acts as a wrapper of components provided by third-party frameworks and it uses the dependency injection pattern to implement recommender systems based on the components selected by the developer.

2. Background

We have created a set of tools that facilitate the process of developing RS called RecoLibry Suite, which architecture is shown in Fig. 1. Firstly, we formalise semantically the representation of components that are typically used in RS, defining their behaviour and restrictions regarding their composition when developing a fully functional RS. This formalisation is carried out through an ontology called RecOnto. The second tool is RecoLibry-studio. It is a web application that guides the design process of an RS using RecOnto. The components described semantically by the ontology have their corresponding implementation in RecoLibry-core. It provides the components defined in RecOnto by wrapping external frameworks such as Mahout [5], Lenskit [3] or jCOLIBRI [4]. In addition, RecoLibry-core allows to use dependency injection to easily compose recommender systems from the components provided. Finally, RecoServer tool automatically deploys the RS

[*] Corresponding author.
E-mail addresses: jljorro@ucm.es (J.L. Jorro-Aragoneses), jarecio@ucm.es (J.A. Recio-García), belend@ucm.es (B. Díaz-Agudo), gjimenez@ucm.es (G. Jimenez-Díaz).

https://doi.org/10.1016/j.knosys.2019.07.025
0950-7051/© 2019 Elsevier B.V. All rights reserved.

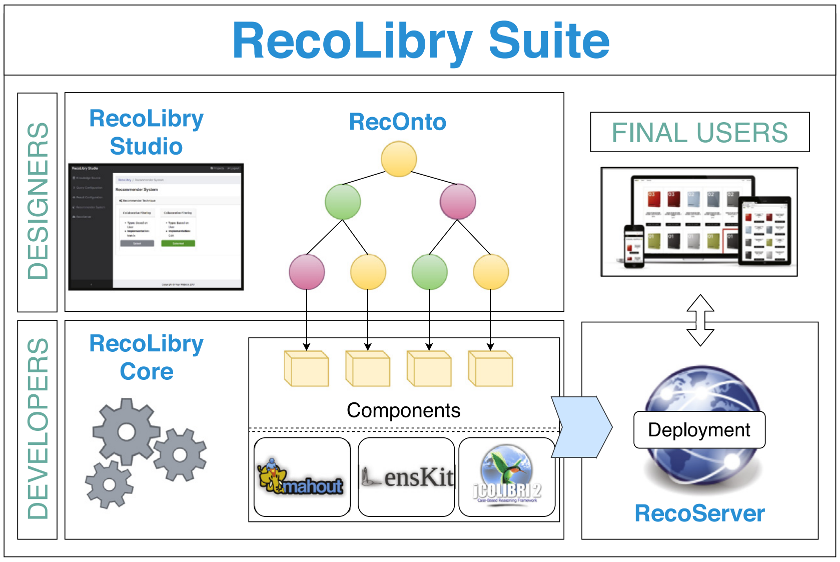

Fig. 1. RecoLibry Suite software architecture.

Table 1
Software metadata.

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	0.0.4
S2	Permanent link to executables of this version	https://github.com/UCM-GAIA/RecoLibry-Core
S3	Legal Software License	GNU v3.0
S4	Computing platform/Operating system	Microsoft Windows, Mac OSx, Linux
S5	Installation requirements & dependencies	Java JDK 8, Apache Maven
S6	If available, link to user manual — if formally published include a reference to the publication in the reference list	https://github.com/UCM-GAIA/RecoLibry-Core/wiki/Home-English
S7	Support email for questions	jljorro@ucm.es

and creates a RestFull-API with all the RS functionalities. It allows to use this API in an external application.

In the following section, we describe how the components are implemented in RecoLibry-core and how it uses dependency injection to build the final deployed RS.

3. Software framework

RecoLibry-core uses a scheme based on the definition of the minimum set of components necessary to build a new recommender system. The most important component in this scheme is `RecommenderSystem`. A `RecommenderSystem` component specifies the functionality of a RS. It needs two elements. The first one is a component that implements the `RecommenderAlgorithm` interface. It defines the methods that an algorithm must implement to be integrated in RecoLibry-core. The second one is an object that implements the `Query` interface. The goal of this interface is to define the structure of the queries that can be used in the implemented RS. Finally, `RecommenderAlgorithm` returns a list of `RecommenderResult` objects.

Regarding the concrete implementations provided by the framework, current version includes components to develop content-based RS, implemented through the jCOLIBRI framework, and collaborative-filtering RS, implemented by the Mahout library.

In addition, RecoLibry-core proposes a development process based on the injection dependency pattern. Concretely, it uses Google's Guice library [9] to easily compose the components included in the framework. In a nutshell, it adds a set of Java annotations that define how the components have to be combined. For example, Listing 1 shows the annotation of the `RecommenderSystem` component through the `@Inject` tag. This way, we define that a `RecommenderAlgorithm` and a `Query` objects are required to build a `RecommenderSystem`.

Following the dependency injection pattern, developers can easily build a RS by extending the `RecSysConfiguration` class or by defining a configuration file similar to the following one:

4. Conclusions

In this paper we present RecoLibry-core, a Java framework to create recommendation systems using components. The large number of existing frameworks to create recommendation systems together with the large number of recommendation techniques make the design of these systems very complex. RecoLibry-core alleviates this problem by integrating these frameworks into a homogeneous set of components. This way, RecoLibry-core provides the required components to build RS and defines a composition process through the dependency injection design pattern that eases the development of this type of systems.

Currently, RecoLibry-core provides components to build classic RS such as collaborative filtering and content-based. In future versions of RecoLibry-core we will add additional features such

Listing 1: Definition of RecommenderSystem constructor.

```
public class RecommenderSystem {
    @Inject
    public RecommenderSystem(RecommenderAlgorithm  algorithm,
                             Query query) {...}
}
```

Listing 2: Composition of recommender system with a JSON file.

```
{"injections": [{
 "type": "Class",
 "bind": "es.ucm.fdi.gaia.recolibry.api.RecommenderAlgorithm",
 "to": "es.ucm.fdi.gaia.recolibry.impl.MatrixFactorization"
},{
 "type": "Class",
 "bind": "es.ucm.fdi.gaia.recolibry.api.Query",
 "to": "es.ucm.fdi.gaia.recolibry.impl.MFQuery"
}]}
```

Table 2
Code metadata.

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.0.4
C2	Permanent link to code/repository used of this code version	https://github.com/UCM-GAIA/RecoLibry-Core
C3	Legal Code License	GNU v3.0
C4	Code versioning system used	Git
C5	Software code languages, tools, and services used	Java 8
C6	Compilation requirements, operating environments & dependencies	Apache Maven
C7	If available link to developer documentation/manual	https://github.com/UCM-GAIA/RecoLibry-Core/wiki/Home-English
C8	Support email for questions	jljorro@ucm.es

as explanations or group-based and context-aware recommendations.

Acknowledgements

Supported by the UCM (Research Group 921330), the Ministry of Economy and Competitiveness (TIN2017-87330-R) and the funding provided by Banco Santander in UCM (CT17/17-CT17/18).

Appendix. Required metadata

Current executable software version

See Table 1.

Current code version

See Table 2.

References

[1] F. Ricci, L. Rokach, B. Shapira, Recommender systems: introduction and challenges, in: Recommender Systems Handbook, Springer, 2015, pp. 1–34.
[2] D. Paraschakis, B.J. Nilsson, J. Holländer, Comparative evaluation of top-n recommenders in e-commerce: An industrial perspective, in: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015, pp. 1024–1031.
[3] M.D. Ekstrand, M. Ludwig, J.A. Konstan, J.T. Riedl, Rethinking the recommender research ecosystem: Reproducibility, openness, and Lenskit, in: Proceedings of the Fifth ACM Conference on Recommender Systems - RecSys '11, ACM Press, New York, New York, USA, 2011, p. 133.
[4] J.A. Recio-García, B. Díaz-Agudo, P.A. González-Calero, Prototyping recommender systems in jcolibri, in: Proceedings of the 2008 ACM Conference on Recommender Systems - RecSys '08, ACM Press, New York, New York, USA, 2008, p. 243.
[5] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning Publications Co., 2011, p. 375, Online, doi:citeulike-article-id:7544201, URL http://www.manning.com/owen/.
[6] J. Kirk, Tensorrec, 2018, https://github.com/jfkirk/tensorrec.
[7] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: An open-source software for multi-class imbalance learning, Knowl.-Based Syst. 174 (2019) 137–143.
[8] J.M. Moyano, E.L. Gibaja, S. Ventura, MLDA: A tool for analyzing multi-label datasets, Knowl.-Based Syst. 121 (2017) 1–3.
[9] R. Vanbrabant, Google Guice: Agile Lightweight Dependency Injection Framework, APress, 2008.