future internet

MDPI

Article

A Multi-Agent Architecture for Data Analysis

Gianfranco Lombardo, Paolo Fornacciari, Monica Mordonini, Michele Tomaiuolo and Agostino Poggi *

Department of Engineering and Architecture, University of Parma, 43124 Parma, Italy; gianfranco.lombardo@unipr.it (G.L.); paolo.fornacciari@unipr.it (P.F.); monica.mordonini@unipr.it (M.M.); michele.tomaiuolo@unipr.it (M.T.)

* Correspondence: agostino.poggi@unipr.it; Tel.: +39-0521-905728

Received: 20 December 2018; Accepted: 13 February 2019; Published: 18 February 2019

Abstract: ActoDatA (Actor Data Analysis) is an actor-based software library for the development of distributed data mining applications. It provides a multi-agent architecture with a set of predefined and configurable agents performing the typical tasks of data mining applications. In particular, its architecture can manage different users' applications; it maintains a high level of execution quality by distributing the agents of the applications on a dynamic set of computational nodes. Moreover, it provides reports about the analysis results and the collected data, which can be accessed through either a web browser or a dedicated mobile APP. After an introduction about the actor model and the software framework used for implementing the software library, this article underlines the main features of ActoDatA and presents its experimentation in some well-known data analysis domains.

Keywords: social media analysis; data mining; multi-agent architecture; actor model; Java

1. Introduction

In the last few years, there has been a growing interest in managing and collecting large, heterogeneous amounts of data concerning people, their habits, and opinions. This interest is mainly due to the new kinds of interactions that have been enabled by the widespread use of mobile technologies and social media platforms by an ever-increasing number of people [1]. As a consequence of this social phenomenon, the interest in analyzing these data is shared among a varied set of stakeholders, who aim at extracting useful information from them. For example, while companies are mainly interested in information about fidelity to their brands and products, politicians and decision-makers look for information about emerging social trends, political opinions, and, more recently, about the diffusion of fake news. Another consequence is that, quite recently, several research groups and information technology companies have worked and are still working on providing data analysis tools and libraries and so, nowadays, developers have several alternatives for building their data analysis applications.

This article presents an actor-based software library, ActoDatA (Actor Data Analysis), providing a set of suitable software components for realizing a flexible infrastructure for data mining applications. It can integrate data analysis and machine-learning algorithms and components provided by the different Java and Python software libraries available on the market. The next section provides an overview of the software framework used for the implementation of ActoDatA. Section 3 describes the multi-agent architecture and the other software components offered by ActoDatA for the development of social media applications. Section 4 shows the experimentation of the software library in some well-known data analysis application domains. Section 5 introduces related work. Finally, Section 6 concludes the article by discussing its main features and the directions for future work.

2. ActoDeS Overview

ActoDeS (Actor Development System) is a software framework for the development of concurrent and distributed systems [2,3], implemented by using the Java language and experimented in different application domains (see, for example, [4–6]). This software framework takes advantage of concurrent objects (hereafter named actors) derived from the actor model [7], and of pre-existent Java software libraries and solutions for supporting concurrency and distribution. The development of a standalone or distributed application consists in (*i*) the definition of the behaviors assumed by its actors and (*ii*) the configuration of a few parameters. Through appropriate configuration, it is possible to select the most appropriate implementation of the actors and of the other runtime components of the application. In fact, the possibility of using different implementations of the actors and of the other runtime components can guarantee some execution attributes of an application (e.g., performance, reliability, and scalability). Moreover, the deployment of an application on a distributed architecture is simplified because an actor can move to another computational node and can transparently communicate with remote actors.

In ActoDeS, an application is defined by a set of actors that interact by exchanging asynchronous messages. Actors can send messages only to known actors, i.e., actors of which they know the addresses. There are three ways for an actor to know the address of another actor: (*i*) It receives a message from the other actor; (*ii*) it receives a message whose content contains the address of the other actor; and finally (*iii*) it creates the other actor. After their creation, actors can change their behaviors several times, until they kill themselves.

Each behavior has the main duty of processing the incoming messages that match certain message patterns. Therefore, if an unexpected message arrives, the actor maintains it in its mailbox until a behavior will be able to process it, or a behavior kills the actor. The processing of the incoming messages is performed through message handlers. In response to a message, an actor uses the associated handler for: (*i*) Sending other messages; (*ii*) creating new actors; (*iii*) changing its local state or its behavior; (*iv*) setting a timeout within receiving a new message; and finally (*v*) killing itself. In particular, when a timeout fires, the actor automatically sends a timeout message to itself and the corresponding message handler is executed.

In ActoDeS, addresses, messages, and message patterns are all instances of Java classes. An actor address is represented by a reference object that acts as both a system-wide unique identifier and a local and remote proxy of an actor. A message is an object that contains a set of fields, maintaining the typical header information and the message content. Finally, a message pattern is an object defined as a combination of constraints on the value of some message fields.

Depending on the complexity of the application and on the availability of computing and communication resources, a system can be distributed on one or more actor spaces. Each actor space corresponds to a Java Virtual Machine and so a system distributed on some actor spaces can be deployed on one or more computational nodes. An actor space acts as "container" for a set of actors and provides them the necessary services for their execution. Figure 1 shows the structure of an actor space.

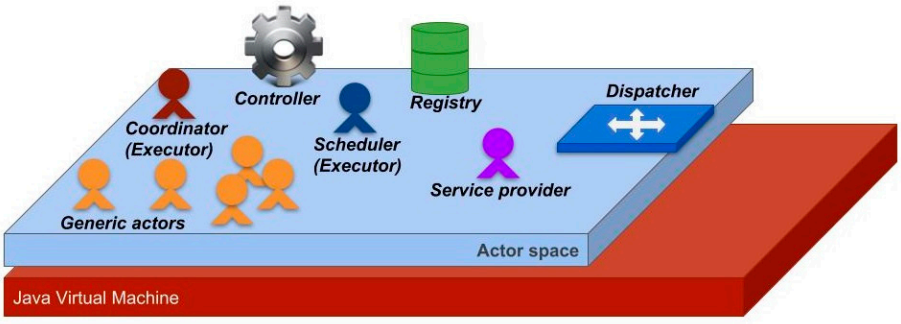

Figure 1. The actor space structure.

In particular, an actor space performs its tasks through three main runtime components (i.e., controller, dispatcher and registry) and through two runtime actors (i.e., the executor and the service provider).

The controller is a runtime component that has the duty to manage the activities of an actor space. It configures and starts the runtime components and the actors of the application and manages its activities until the end of its execution.

The dispatcher is a runtime component that is meant to support the communication with the other actor spaces of the application. It creates connections to/from the other actor spaces, manages the reception of messages from the input connections, and delivers messages through the output connections. This component can be configured by using different communication technologies (the current release of the software supports ActiveMQ, MINA RabbitMQ, RMI, and ZeroMQ). Therefore, an application can be configured with one of the such communication technologies. It can be useful to support the interaction of an ActoDatA with other applications. Morever, ZeroMQ is used internally in an ActoDatA application to support the communication between Java and Python agents.

The registry is a runtime component that supports actor creation and message passing. In fact, it creates the references of new actors and supports the delivery of those messages that come from remote actors by mapping each remote reference onto the local reference to the final destination of the message.

The executor is a runtime actor that manages the execution of the actors of an actor space and may create its initial actor(s). Of course, the duties of an executor depend on the type of implementation of its actors and, in particular, on the type of threading solutions used in their implementation. In fact, an actor can have its own thread (hereafter named active actor), or it can share a single thread with the other actors of the actor space (hereafter named passive actor). Hence, while the executors of active actors (called coordinators) delegate a large part of their work to the Java runtime environment, the executors of passive actors (called schedulers) must completely manage their execution by initializing them and by cyclically calling the method that performs a step of their execution.

The service provider is a runtime actor that offers, to the other actors of the application, the possibility of performing new kinds of actions (e.g., to broadcast or multicast a message and to move from one actor space to another). In fact, it provides other actors with a set of additional services that they can employ by sending a message to the service provider.

As introduced above, ActoDeS supports the interaction of its (Java) actors with some Python actors that provide a similar behavior to that of the Java actors. It was done by defining a Java and a Python broker that support (*i*) the discovery of the "foreign" actors, (*ii*) the exchange of JSON messages between the Java and Python actors, and finally (*iii*) the conversion of the JSON object in either the Java or Python objects and vice versa.

3. ActoDatA Overview

The features of the actor model and the flexibility of its implementation make ActoDeS suitable for building agent-based data analysis applications. In particular, actors have the suitable features for defining the agent models found in MAS and DAI systems [8]. In fact, actors and computational agents share certain characteristics: (*i*) Both react to external stimuli (i.e., they are reactive); (*ii*) both are self-contained, self-regulating, and self-directed (i.e., they are autonomous); and (*iii*) both interact through asynchronous messages, with such messages being the basis for their coordination and cooperation (i.e., they are social). Therefore, an actor library can be considered a suitable solution for the implementation of a multi-agent system. In our case, the actors provided by ActoDeS cannot be considered real agents, but the different types of actors defined and implemented for the development of ActoDatA applications offer some features for which they can be considered agents.

Moreover, the use of ActoDeS simplifies the development of distributed, flexible, and scalable data analysis applications. In fact, the use of active and passive actors allows the development of

applications involving a large number of actors, where each actor can perform a different behavior and react to the input data (i.e., messages) in real time.

However, ActoDeS does not offer specific components for building data analysis applications. Therefore, we defined a software library, called ActoDatA (Actor Data Analysis), that, starting from ActoDeS, provides a set of agents and a multi-agent architecture useful for the development of such kinds of applications. The current implementation of ActoDatA supports the analysis of email and Twitter data. However, it can be easily extended to support the analysis of other types of data.

In particular, ActoDatA provides a multi-agent architecture which eases the deployment and the concurrent execution of several social media analysis applications. This architecture takes advantage of: (*i*) A set of agents that perform all the analysis tasks; (*ii*) a data store that maintains data sets and analysis results; (*iii*) a user console that allows developers to configure, start, stop, resume, and kill applications; and (*iv*) a web/APP server that presents information about collected data and analysis results. Figure 2 shows the architecture of the ActoDatA multi-agent system.

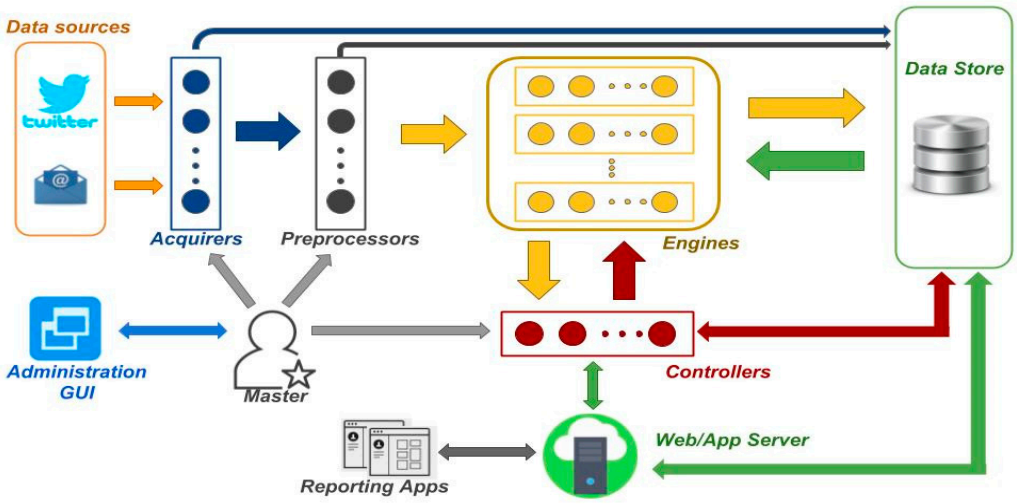

Figure 2. The ActoDatA (Actor Data Analysis) multi-agent architecture.

ActoData takes advantage of five types of agent: ***Acquirer***, ***preprocessor***, ***engine***, ***controller***, and ***master***. Each agent acts as a wrapper for predefined or custom components that perform the different data analysis tasks of an application. For each type of agent there are some implementations suitable to wrap some specific data analysis components. In particular, each application involves some ***acquirer***, ***preprocessor***, and ***engine*** agents and a *controller* agent. Moreover, a ***master*** agent has the duty of deploying and monitoring the execution of all the applications of a multi-agent architecture.

An ***acquirer*** agent has the duty of managing a data source, acquiring the data, and submitting some or all of the data to the agents initiating their processing. This agent is specialized for a specific data source and performs iteratively the following activities:

- Waits for new data
- Filters the data on the basis of a specific pattern or a set of constraints (optional)
- Sends the (filtered) data to the registered preprocessor agents.

As introduced above, for each type of agent there are different implementations. For this type of agent there are two implementations. The first supports the filtering of Twitter messages (in this case, filtering is based on the Twitter API and text pattern matching) and the second supports the filtering of email messages (in this case, filtering is based on the JavaMail API and text pattern matching). Figure 3 shows the code of a filter selecting Twitter messages on the basis of their language and the presence of keywords.

```
cb.setDebugEnabled(true)
  .setJSONStoreEnabled(true)
  .setOAuthConsumerKey(g.getString("api.key"))
  .setOAuthConsumerSecret(g.getString("api.secret"))
  .setOAuthAccessToken(g.getString("token.key"))
  .setOAuthAccessTokenSecret(g.getString("token.secret"));

this.filter  = new FilterQuery();

String[] tk = g.getString("keywords").split(",");

if (tk != null)
{
  this.filter.track(tk);
}

String[] tl = g.getString("languages").split(",");

if (tl != null)
{
  this.filter.language(tl);
}

this.reference = r;

this.running = false;

this.twitterStream = new TwitterStreamFactory(cb.build()).getInstance();
```

Figure 3. The code of a filter for Twitter messages selecting messages on the basis of the language and the presence of keywords.

A ***preprocessor*** agent has the duty of performing the typical preprocessing operations (i.e., cleaning, instance selection, normalization, transformation, and feature extraction [9]) on the data coming from the acquirer agents, to which it registered its interest to receive the data. This agent can perform different tasks by encapsulating different preprocessing modules that can be implemented in two different languages, Java and Python. This agent performs iteratively the following activities:

- Processes the data by using the embedded preprocessing module
- Sends the data to the registered engine agents.

An ***engine*** agent has the duty of performing the classification of the data coming the ***preprocessor*** agents, to which it registered its interest to receive the data. This agent embeds a classifier (usually, a Weka [10] or a scikit-learn [11] classifier) that can be implemented either in Java or Python. Moreover, it must have access to a training and a test set and be able to update the training set by using the incoming data. After the reception of a fixed number of new instances, this agent performs the following activities:

- Runs the embedded classifier on a sequence of data sets
- Sends the result sets to the controller and updates the training sets (optional)
- Usually, a new iteration starts when the engine agent receives a certain number of new data.

A ***controller*** agent has the duty of managing one or more engines working on the same data sets. This agent performs iteratively the following activities:

- Compares their results and saves the results on the data storage
- Notifies the web/APP server about activities and results

Finally, a ***master*** agent has the duty of managing the applications of a specific multi-agent architecture. In particular, it performs the following activities:

- Interacts with the users through the specific console
- Monitors the execution of the applications, logging their relevant events
- Notifies those events to the web/APP server and to the user console.

Users can view the analysis results through a web browser and a reporting APP. In particular, both of the interfaces show information about the analysis results and collected data and update the presentation of the data dynamically. In particular, such interfaces can show:

- Pie and line charts presenting, for example, the data that have eventually been classified for each class during the observed period
- The total number of instances, repeated data, and discarded data
- The numeric results of classifiers for all the data
- The frequency of classifications of the latest data.

As introduced above, ActoDatA supports the analysis of email and Twitter data. This is done through a mailing and a Twitter service. The mailing service accepts requests from actors for registering their interest to receive the emails sent from a set of addresses, receives incoming messages into a specific mailbox, and forwards them to subscriber actors. The Twitter service allows actors to register their interest on some tweets from the widespread Twitter social platform. In particular, it allows actors to send the following kinds of requests:

- User timeline, to obtain the recent tweets of a specified user and save them in a local storage system
- Content query, to obtain recent tweets published on Twitter and selected according to constraints specified by the actor
- Stream, to receive messages published on the platform during the execution. Tweets are obtained in the form of JSON objects.

The development of an application mainly consists in the definition of a configuration file where the following are defined: (*i*) The agents involved in the application; (*ii*) the relationships among them; and (*iii*) for each agent the specific data analysis components used in their activities (e.g., filters, classifiers and training and test sets). Therefore, an application can easily modify the data analysis components and the number of actors. Of course, the current implementation of ActoDatA supports only the analysis of email and Twitter data and Weka and scikit-learn classifiers, but its future evolution should provide a high flexibility in the development of applications.

4. Experimentation

We experimented, and are experimenting, with the use of the proposed multi-agent architecture in order to perform different types of data analysis, particularly in the field of social media analysis [12]. In the following, we describe several examples of applications in order to highlight the flexibility that ActoDatA enables in the design and implementation of different kinds of data analysis applications starting from different data sources. For the sake of completeness, we report also some context-dependent results of each application for a better understanding of the reader.

The most relevant experimentations are related to tweet topic detection [13], troll detection [14], an image-based system for hoax detection [15], and a sort of automatic chatbot for citizens' reports in the field of the e-government [16].

4.1. Tweet Topic Detection

In the tweet topic classification, we used "sport" as general topic (i.e., tweets with the #sport hashtag) because it is a widespread topic for which it is easy to find an abundance of data and news. In particular, we decided (*i*) to classify tweets among a few chosen sports (i.e., soccer, rugby, basketball, and tennis) and (*ii*) to compare the results of different classifiers and different classification processes.

In this case, the media analysis application involved an acquirer, a preprocessor agent, and some engine agents. One of the most interesting results of the experimentation was that if a classifier used its own classification results to augment its own training set continuously, then its classification quality increased. We obtained such a result with an experimentation involving two engines with the same classifier and only one of them updated its training set. Both classifiers were tested against the same test set, composed of 400 instances and distributed almost equally among the various classes. Moreover, the classification quality is re-evaluated automatically after the reception of 30 tweets. While the engine not updating its training set has shown a level of classification accuracy equal to 62.5%, the engine updating its training set increased its accuracy from 62.5% up to 76.0% after collecting 1050 tweets.

4.2. TrollPacifier

Starting from the ActoDatA architecture, we have developed TrollPacifier, a holistic system for troll detection, which analyzes many different features of trolls and legitimizes users on the popular Twitter platform. In particular, the identification of trolls is done by taking advantage of the six most important types of features proposed in literature: Sentiment analysis features [17], time and frequency of actions features, text content and style, user behaviors [18], interactions with the community [19], and advertisement of external content [20]. TrollPacifier contains a set of preprocessor and engine agents dedicated to support the different types of approaches for troll detection. Figure 4 shows the structure of the TrollPacifier multi-agent system. The experimentation involved the definition of a training set containing 500 trolls and 500 non-troll users. The results of the experimentation were: (*i*) A high accuracy in the identification of trolls (95.5%) and (*ii*) a demonstration of how the ActoDatA architecture can support an automatic classification involving several agents and several types of tasks (e.g., acquiring streaming data and users' profile information from Twitter, performing feature extraction tasks and performing classification based on different machine learning algorithms).

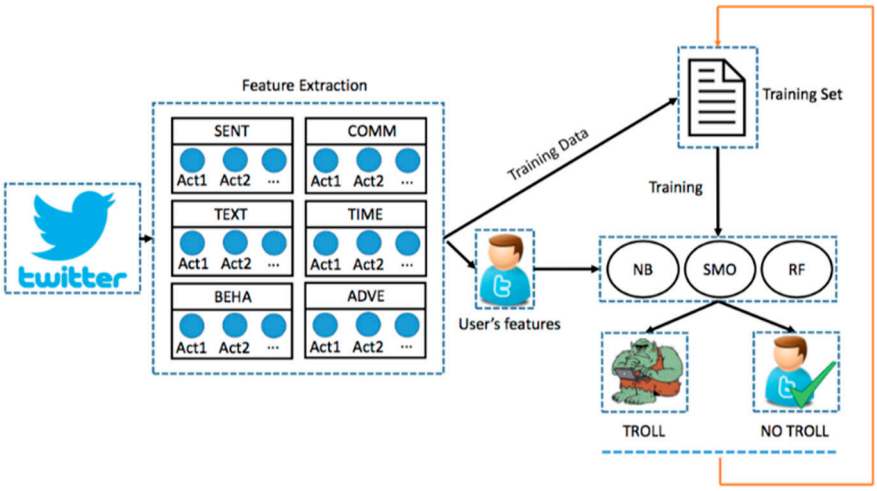

Figure 4. The TrollPacifier multi-agent system.

4.3. Image Based Hoax Detection

As part of a more comprehensive project for the detection of fake news on social media, we have developed a specific system with ActoDatA. It retrieves the images associated with on-media shared content and tries to evaluate their trustworthiness by extracting some visual features and training different kinds of classifiers. The training and test datasets were collected by using two acquirer actors for Facebook. The first acquirer collected data from public, fake Facebook pages, and the other one from trusted and institutional pages. At the end, more than 1000 images were retrieved for building an appropriate dataset. Then, we defined a preprocessor agent based on Google Vision API to extract

visual features from images. For each image, various kinds of features were considered, including its color composition, the recognized objects, and the list of sites on which it was published. These features were then used for training different classifiers (in the form of ActoData engine agents) using different subsets of features and different machine learning algorithms (Bernoulli NB, Logistic Regression, Decision Trees, Random Forest and Support Vector Machine). This approach allowed us to reach an overall accuracy of 83% in hoax identification using the well-known Bernoulli NB algorithm.

4.4. Automatic Processing and Classification of Citizens' Reports

In collaboration with the Montecchio Emilia Municipality, we developed a comprehensive framework for managing reports sent to the local government by citizens through well-known instant messaging apps (WhatsApp, Telegram and Facebook Messenger). It leveraged a combined use of web systems and automated bots and, using machine learning techniques, it understood what kind of report the citizens wanted to send to the municipality from the received text and the associated image. Reports are automatically categorized in one of four predefined classes (environment, lighting, maintenance, and security), by a classification system based on text-analysis, image-recognition and object-detection. In this case we used three different acquirers that work as chatbots for report sources. Then, different pre-processors extracted features for the next text classifier and image classifier to create two different bag-of-words models (BoW). At the end, a unique classifier (engine actor) using a joint BoW model was trained for automatic categorization. To manage the entire system, we developed a web-based front-end in the form of an ad-hoc customer relationship management (CRM) system. Figure 5 shows the data flow describing the management of the citizen reports.

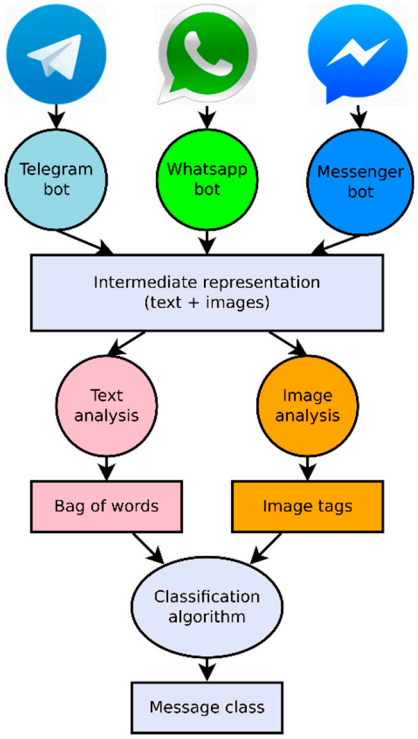

Figure 5. The data flow describing the citizen reports management.

5. Related Work

The focus of social network analysis is on relationships among the members of social communities. This analysis is fundamental in the social and political sciences, as well as in economics and industrial engineering. Historically, social network analysis is a research strategy that allows researchers to quantify the pattern of relations among a set of actors. A comprehensive introduction to the theory

and practice of network analysis in the social sciences is provided in [12]. In addition to the structural analysis of a social entity, it is interesting to study the emotional components of community members. This is a research field that takes the name of opinion mining and sentiment analysis. The growing availability and popularity of online social media and personal blogs gives new opportunities and challenges in the use of information technologies and machine learning to provide tools to understand the opinions of others.

A survey of sentiment analysis and opinion mining techniques is shown in [21]. Examples of applications of sentiment analysis are shown in [22], where a corpus of geotagged tweets was used for estimating happiness in Italian cities, or in [23], where the US mood throughout the day was inferred from an analysis performed on Twitter data. In [24], the authors present a possible combined approach between social network analysis and sentiment analysis. In particular, a sentiment is associated with the nodes of the graphs showing the social connections, which may highlight potential correlations. The growth and pervasiveness of online communities allow different and new possibilities to interact with them. For example, social networking systems blur the distinction between the private and working spheres and users are known to use such systems both at home and in the work place, both professionally and with recreational goals [25].

Moreover, social networking has a large potential for also easing collaboration across organizational boundaries, but effective e-collaboration through social networks requires the development of open and interoperable systems. This could create some difficulties in the domains of privacy or confidentiality [4]. Some work has been done in the analysis of undesirable behavior such as trolling [26]. Like other organizations, sport industry groups including athletes, teams, and leagues use social media to promote and share information about their products. In [27], authors explored how sporting event organizers and influential Twitter users spread information through the online social network.

In general, data coming from online social media is an example of big data, which often requires the use of distributed and scalable software tools. Several actor-oriented libraries have been proposed in recent years, which could be used to implement a distributed sentiment analysis system (see, for example, Scala [28] and Akka [29]). Scala is an object-oriented and functional programming language that provides an implementation of the actor model, unifying thread-based and event-based programming models. In fact, in Scala an actor can suspend with a full thread stack (receive) or can suspend with just a continuation closure (react). Therefore, scalability can be obtained by sacrificing program simplicity. Akka is an alternative toolkit and runtime system for developing event-based actors in Scala, but also providing APIs for developing actor-based systems in Java. One of its distinguishing features is the hierarchical organization of actors. In fact, a parent actor that creates children actors is also responsible for handling their failure.

The current implementation of ActoDatA is based on the ActoDeS software framework, however, the porting on other actor or agent-based software libraries should be quite easy. This is certainly true for Scala and Akka because, in the same way as ActoDeS, they support the communication through the use of ZeroMQ and so they can use it to exchange messages with Python agents. The main difference between ActoDeS and Scala and Akka is that ActoDeS provides heavyweight actors, while Scala and Akka provide lightweight actors. Therefore, while ActoDeS actors can take advantage of a set of predefined high-level services, easing the development of applications, Scala and Akka actors offer a few low-level services, which may complicate the development of applications, but may offer better performances than ActoDeS applications.

6. Conclusions

In this work, we have proposed the software library ActoDatA for the development of distributed data mining applications. The library provides a multi-agent architecture, where a set of agents perform the main tasks required in the data mining domain.

We experimented with ActoDatA in different types of data analysis applications. Its use simplified the development of such applications with respect to the tools we previously used. The main advantage of ActoDatA is the possibility to use all the software components provided by the different data analysis and machine learning software libraries available in Java and Python. In fact, the use of a new data analysis component implies the development of a custom agent that wraps it; usually this operation has short length and low complexity. Moreover, the possibility to distribute the computation on several nodes allowed us to run several applications concurrently, maintaining a good level of execution quality.

The current work has the goal of continuing the experimentation with ActoDatA. Moreover, future work will be dedicated to improving the software library with the support of different data sources and guaranteeing a good fault tolerance [30]. For achieving this last goal, we are planning to use hardware redundancy and some multi-agent fault-tolerance coordination techniques. Moreover, we are planning to extend the use of ActoDatA to the analysis of the information coming from the structure of social networks [31,32]. Finally, we are planning the porting of ActoDatA to the JADE software framework [33].

Author Contributions: Experimentation and conceptualization, M.M., M.T., and A.P.; data curation, P.F.; software, A.P.; writing—original draft, A.P.; writing—review & editing, G.L. and M.T.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Franchi, E.; Poggi, A.; Tomaiuolo, M. Social Media for Online Collaboration in Firms and Organizations. *Int. J. Inf. Syst. Model. Des.* **2016**, *7*, 18–31. [CrossRef]
2. Bergenti, F.; Poggi, A.; Tomaiuolo, M. An Actor Based Software Framework for Scalable Applications. In *Internet and Distributed Computing Systems, IDCS 2014, Lecture Notes in Computer Science*; Springer Nature: Cham, Switzerland, 2014; Volume 8729, pp. 26–35.
3. Bergenti, F.; Iotti, E.; Tomaiuolo, M.; Poggi, A.; Mastorakis, N.; Mladenov, V.; Bulucea, A. Concurrent and Distributed Applications with ActoDeS. In Proceedings of the 20th International Conference on Circuits, Systems, Communications and Computers (CSCC 2016), Corfu Island, Greece, 14–17 July 2016.
4. Franchi, E.; Poggi, A.; Tomaiuolo, M. Blogracy: A Peer-to-Peer Social Network. *Int. J. Distrib. Syst. Tech.* **2016**, *7*, 37–56. [CrossRef]
5. Angiani, G.; Fornacciari, P.; Lombardo, G.; Poggi, A.; Tomaiuolo, M. Actors Based Agent Modelling and Simulation. In *Communications in Computer and Information Science*; Springer Nature: Cham, Switzerland, 2018.
6. Fornacciari, P.; Lombardo, G.; Mordonini, M.; Poggi, A.; Tomaiuolo, M. Agent Based Cellular Automata Simulation. In Proceedings of the 19th Workshop "From Objects to Agents", WOA 2018, Palermo, Italy, 28–29 June 2018; Volume 2215, pp. 16–20.
7. Agha, G.A. *ACTORS: A Model of Concurrent Computation in Distributed Systems*; MIT Press: Cambridge, MA, USA, 1985.
8. Kafura, D.; Briot, J.P. Actors and agents. *IEEE Concurr.* **1998**, *2*, 24–28. [CrossRef]
9. Cagnoni, S.; Fornacciari, P.; Kavaja, J.; Mordonini, M.; Poggi, A.; Solimeo, A.; Tomaiuolo, M. Automatic Creation of a Large and Polished Training Set for Sentiment Analysis on Twitter. In *Machine Learning, Optimization, and Big Data, MOD 2017, Lecture Notes in Computer Science*; Springer Nature: Cham, Switzerland, 2017; Volume 10710, pp. 146–157.
10. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.* **2009**, *11*, 10–18. [CrossRef]
11. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-Learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2012**, *12*, 2825–2830.
12. Scott, J. *Social Network Analysis*; Sage: London, UK, 2012.

13. Fornacciari, P.; Mordonini, M.; Poggi, A.; Tomaiuolo, M. Software Actors for Continuous Social Media Analysis. In Proceedings of the 19th Workshop "From Objects to Agents", WOA 2017, Scilla, Italy, 15–16 June 2017; Volume 1867, pp. 84–89.
14. Fornacciari, P.; Mordonini, M.; Poggi, A.; Sani, L.; Tomaiuolo, M. A holistic system for troll detection on Twitter. *Comput. Hum. Behav.* **2018**, *89*, 258–268. [CrossRef]
15. Angiani, G.; Balba, G.; Fornacciari, P.; Lombardo, G.; Mordonini, M.; Tomaiuolo, M. Image-Based Hoax Detection. In Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good, Bologna, Italy, 28–30 November 2018.
16. Angiani, G.; Fornacciari, P.; Lombardo, G.; Mordonini, M.; Pietroni, U.; Tomaiuolo, M. Automatic processing and classification of citizens' reports. In Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good, Bologna, Italy, 28–30 November 2018; pp. 310–311.
17. Cambria, E.; Poria, S.; Hazarika, D.; Kwok, K. Senticnet 5: Discovering conceptual primitives for sentiment analysis by means of context embeddings. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligen, New Orleans, LA, USA, 2–7 February 2018; pp. 1795–1802.
18. Cheng, J.; Bernstein, M.; Danescu-Niculescu-Mizil, C.; Leskovec, J. Anyone Can Become a Troll: Causes of Trolling Behavior in Online Discussions. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education-ITiCSE '17, Portland, OR, USA, 25 February–1 March 2017; Volume 2017, pp. 1217–1230.
19. Kumar, S.; Spezzano, F.; Subrahmanian, V. Accurately detecting trolls in Slashdot Zoo via decluttering. In Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014), Beijing, China, 17–20 August 2014; pp. 188–195.
20. Aro, J. The Cyberspace War: Propaganda and Trolling as Warfare Tools. *Eur. View* **2016**, *15*, 121–132. [CrossRef]
21. Liu, B. Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies. Available online: https://doi.org/10.2200/S00416ED1V01Y201204HLT016 (accessed on 14 February 2019).
22. Allisio, L.; Mussa, V.; Bosco, C.; Patti, V.; Ruffo, G. Felicittà: Visualizing and estimating happiness in Italian cities from geotagged tweets. In Proceedings of the 1st International Workshop on Emotion and Sentiment in Social and Expressive Media: Approaches and Perspectives from AI, Turin, Italy, 3 December 2013; Volume 1096, pp. 95–106.
23. Mislove, A.; Lehmann, S.; Ahn, Y.Y.; Onnela, J.P.; Rosenquist, J.N. Pulse of the Nation: U.S. Mood throughout the Day Inferred from Twitter. Available online: http://www.ccs.neu.edu/home/amislove/Twittermood/ (accessed on 18 December 2018).
24. Fornacciari, P.; Mordonini, M.; Tomaiuolo, M. Social network and sentiment analysis on Twitter: Towards a combined approach. In Proceedings of the 1st International Workshop on Knowledge Discovery on the WEB, Cagliari, Italy, 3–5 September 2015; Volume 1489, pp. 53–64.
25. Angiani, G.; Fornacciari, P.; Mordonini, M.; Tomaiuolo, M.; Iotti, E. Models of Participation in Social Networks. In *Advances in Social Networking and Online Communities*; IGI Global: Hershey, PA, USA, 2017; pp. 196–224.
26. Cheng, J.; Danescu-Niculescu-Mizil, C.; Leskovec, J. Antisocial Behavior in Online Discussion Communities. In Proceedings of the 9th International AAAI Conference on Web and Social, Oxford, UK, 26–29 May 2015; pp. 61–70.
27. Hambrick, M.E. Six Degrees of Information: Using Social Network Analysis to Explore the Spread of Information within Sport Social Networks. *Int. J. Sport Commun.* **2012**, *5*, 16–34. [CrossRef]
28. Haller, P.; Odersky, M. Scala Actors: Unifying thread-based and event-based programming. *Theor. Comput. Sci.* **2009**, *410*, 202–220. [CrossRef]
29. Rycerz, K.; Bubak, M. Using Akka Actors for Managing Iterations in Multiscale Applications. In *Proceedings of the Parallel Processing and Applied Mathematics, PPAM 2015, Lecture Notes in Computer Science*; Springer Nature: Cham, Switzerland, 2016; Volume 9573, pp. 332–341.
30. Negri, A.; Poggi, A.; Tomaiuolo, M.; Turci, P. Dynamic Grid tasks composition and distribution through agents. *Concurr. Comput. Pract. Exp.* **2006**, *18*, 875–885. [CrossRef]
31. Franchi, E.; Poggi, A.; Tomaiuolo, M. Open Social Networking for Online Collaboration. *Int. J. e-Collab.* **2013**, *9*, 50–68. [CrossRef]

32. Bergenti, F.; Franchi, E.; Poggi, A. Agent-Based Social Networks for Enterprise Collaboration. In Proceedings of the 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Paris, France, 27–29 June 2011; pp. 25–28.
33. Poggi, A.; Tomaiuolo, M.; Turci, P. Extending JADE for agent grid applications. In Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Modena, Italy, 14–16 June 2004; pp. 352–357.