

Architectural Technical Debt of Multiagent Systems Development Platforms

Ilaria Pigazzini¹, Daniela Briola¹ and Francesca Arcelli Fontana¹

¹Department of Informatics, Systems and Communication (DISCO), University of Milano - Bicocca, 20125 Milan, Italy

Abstract

Technical debt is candidate to be the next *buzzword* in software engineering, and the number of studies evaluating the technical debt of software projects is increasing. A particular and dangerous type of debt is the architectural debt, i.e., the consequences of sub-optimal design decisions. Currently, there are no studies about the evaluation of architectural debt in MultiAgent Systems (MAS) and platforms. Hence, in this paper we propose the analysis of four well-known MAS development platforms, with the aim of evaluating their architectural debt and open the discussion in this field. We exploit a tool, named Arcan, developed for architectural smell detection and for the computation of an architectural debt index. The results show that MAS development platforms are subjected to architectural debt, and in particular to the presence of Cyclic Dependency smells. However, there is evidence that the minimum amount of debt is reached when developers report “bug fixes” and “Improvements”.

Keywords

architectural debt, architectural smells, multiagent system platforms, trend analysis

1. Introduction

Technical Debt (TD) is “a metaphor reflecting technical compromises that can yield short-term benefit, but may hurt the long-term health of a software system” [1]. Architectural Technical Debt (ATD) is a specific type of TD limited to the architecture (design decisions) of a software system [1] and is considered as the most dangerous and critical one [2]. Systems affected by ATD are hard to maintain and evolve.

The concept of TD is not recent [3], however the research has been active especially in the past few years. Works on TD and ATD have been done on monolithic systems [4][5], distributed systems such as microservices [6], machine learning systems [7] and also on IoT systems [8]. However, to the best of our knowledge, there are no study about ATD in MultiAgent Systems (MAS) and MAS development platforms. Anyway, the MAS community is deserving to reliability, scalability and in general Software Engineering (SE) aspects more and more attention in the last years, as confirmed for example by the creation of the dedicated SE area of interest at AAMAS (International Conference on Autonomous Agents and Multiagent Systems International Conference on Autonomous Agents and Multiagent Systems), workshops focusing on SE topics (for example EMAS (Engineering Multi-Agent Systems) and AREA (Agents and Robots for reliable Engineered Autonomy) [9]), and works on Engineering MultiAgent


WOA 2021: Workshop “From Objects to Agents”, September 1–3, 2021, Bologna, Italy

✉ i.pigazzini@campus.unimib.it (I. Pigazzini); daniela.briola@unimib.it (D. Briola); i.tiddi@vu.nl (F. A. Fontana)

🆔 0000-0003-2629-6762 (I. Pigazzini); 0000-0003-1994-8929 (D. Briola); 0000-0002-1195-530X (F. A. Fontana)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Systems or surveying in a systematic way the available technologies (for example [10, 11, 12]): so, it is only a matter of time before other SE topics will be faced by the MultiAgent Systems community too.

In this paper, we aim to analyse four well-known and largely adopted MAS development platforms (Jade, Jason, Jadex and Netlogo) in order to evaluate their architectural debt: since these platforms are used by many developers and have been released in many versions in a quite long lifespan, we are interested in evaluating if they suffer of ATD, so that in case to provide to their developers useful hints to improve their quality.

We exploit Arcan [13], our tool for Architectural Smells (AS) detection and ATD estimation. In particular, we compute the Architectural Debt Index (ADI) [4], which is a value indicating the amount of architectural technical debt present in a project, based on the AS that affect it [13]. AS (which by some authors are referred to as anti-patterns) are design decisions which impact negatively on the internal quality of software systems and in this study we consider three different types of AS based on dependency issues.

Our results show that the considered systems suffer from ATD, thus their developers should be aware of it so that to be able to manage these issues in future release.

The outline of our paper is the following: Section 2 reports some related works regarding the quality of MAS platforms; Section 3 describes the study design, with the study research questions and the analysis we conducted to answer them; Section 4 reports the results of our analysis and the answers to the research questions; finally Section 5 contains the conclusions and future developments of our work.

2. Related Works

For the best of our knowledge, this is the first study aiming at evaluating the architectural debt of MAS platforms. In general, we found few references to the quality of MAS platforms. However, even if we did not find studies about their maintainability and evolvability (the two quality attributes most impacted by ATD) we found some works about the security, performance and scalability of MAS platforms. We observed that the quality of the design aspects of MAS platforms and MAS systems are not yet often taken in consideration and discussed [14], even if they have an impact on their final performance, probably because the community is still more focus on the previous mentioned aspects which prevent whatever platform to be concretely and largely adopted.

Concerning security, Endsuleit et al. [15] performed a security analysis on the multiagent platform Jade in its version 3.2 as well as on its security plugin Jade-S. They reported a classification of possible and well-known attacks on Jade and provided a discussion on what is still missing in Jade-S. They also present some Denial-of-Service attacks which they have implemented and successfully tested.

Concerning the performance, Mulet et al. [16] investigated the relationship between performance (in terms of agents' response time to messages) and internal design, that is, to identify the key design decisions that lead to better performance. They measured the performance of three Open-Source MAS platforms, namely Jade, MadKit and AgentScape. They found out that design decisions related to the modularity of the platform, such as offering a message service

by means of agents instead of implementing it in the kernel, degrades performance. Moreover, centralizing services in a single host in the platform also degrades performance because the host can become a bottleneck in the case of very popular services.

A similar study was conducted by Alberola et al. [17], who analysed the same set of three MAS platforms and reached similar conclusions, i.e., that the design impacts the performance. In particular, they evaluated the response time of the three platforms when changing parameters like message traffic and the amount of agents running. They found that all the three platforms perform poorly and demonstrate low scalability when the MAS being run on increases.

To conclude, the field of MAS software quality and technical debt is not popular and researched yet, and with our work we aim to open the discussion about architectural debt and architectural smells by analysing the most used platforms for MAS development.

3. Study Design

We introduce the design of this study and the following Research Questions we aim to answer:

- *RQ1: Which is the most present type of AS in MultiAgents Systems platforms?*
- *RQ2: What can we observe according to architectural debt of MultiAgents Systems platforms?*

To answer the two RQs we evaluate the AD in terms of the AS and the Architectural Debt Index (ADI) computed through Arcan. Since we have large experience [13][4][18] in analyzing the AD in open source projects, but not in MAS development platforms, through the answer to these RQs we aim to analyze the AD of MAS platforms, in order to provide some preliminary hints to their developers. In case AD is present or specific AS are identified in the systems, developers have to pay attention to these problems to prevent them or remove them as soon as possible.

3.1. Analyzed projects

We selected four well-known MAS development platforms, namely Jade [19], Jadex[20], Jason[21] and Netlogo[22], and analysed their development history. These projects are written in Java, the programming language supported by Arcan. All projects but Jade are hosted on Github, which, given the large amount of code commits (code snapshots at specific points in time), enables the easy analysis of their history. Table 1 shows the main project characteristics: names, the number of analysed commits, the considered time period (date of the first and last commit), size expressed in Number of Lines of Code (LOC) both for the first and last commit and finally the download url. Concerning the commit analysis, we considered only commits pushed or merged into the master branch, starting from the beginning of the commit history and by sampling one commit every 30. We do not analyse each commit since architectural changes tend to happen in larger time spans with respect to code changes. A threat to such approach could be that by managing sampling by taking in consideration only the time gap, we would miss out “relevant commits”, where the architectural change actually happens. However, it is not important if we miss relevant commits, because we take into consideration the whole evolution and an

Table 1
Projects characteristics

Project	#Commits	First commit	Last commit	LOC first commit	LOC last commit	Download url
Jade	6	23/12/2015	06/06/2017	-	-	https://mavenrepository.com/artifact/com.tilab.jade/jade
Jadex	111	05/11/2008	16/03/2018	130288	502220	https://github.com/actoron/jadex
Jason	38	23/03/2017	20/04/2021	37447	45825	https://github.com/jason-lang/jason
Netlogo	25	05/08/2011	09/05/2016	60260	56075	https://github.com/NetLogo/NetLogo

architectural change happens during a span (not in a single commit). Similar custom samplings were used in similar context by previous studies [23, 24]. We conducted a different analysis for Jade, which is the only project not hosted on Github. We collected six versions from the Maven Central Repository¹ and run Arcan on all of them. Since we found only the jar files, we could not report the number of Lines of Code in Table 1.

Notice that the selected projects have different history, community, development team and purpose: indeed, different amounts of commits are sampled in each case. This makes the individual results difficult to compare directly. However, we propose a preliminary analysis which shall be complemented with manual validation from developers and additional context information. Section 3.3 offers the details of our analysis and how we compare the results among the different projects.

3.2. Collected Data

An architectural smell (AS) is a software design decision which negatively impact on the system internal quality, e.g., the system maintainability and ability to evolve. We collect data about the presence of AS because they are symptoms of architectural debt. AS can be of different types and have different side-effects.

We describe below the AS detected by Arcan considered in this work:

- *Unstable Dependency (UD)*: describes a component (package) that depends on other sub-systems that are less stable than the component itself. The components with a high instability are more prone to change with respect to the more stable ones, this means that the component which depends on less stable components is forced to change along with them.

¹<https://mavenrepository.com/artifact/com.tilab.jade/jade>

- *Hub-Like Dependency (HL)*: this smell arises when a component (class or package) has (outgoing and ingoing) dependencies with a large number of other components. The component affected by the smell is a unique point of failure and a dependency bottleneck. Moreover the logic inside a Hub-Like Dependency is hard to understand, and the smell causes change ripple effect.
- *Cyclic Dependency (CD)*: refers to a component (class or package) that is involved in a chain of relations that breaks the desirable acyclic nature of a system's dependency structure. The components involved in a CD can be hardly released, maintained or reused in isolation. Moreover, a change on one affected component will propagate towards all the other ones involved in the cycle.

Moreover, through Arcan we are able to compute the Architectural Debt Index of each project, which takes into account: (i) the *Number* of AS detected in a project, (ii) the *Severity* of an AS, where for Severity we mean the criticality of each instance of AS (an instance of a type of smell, such as CD, can be more critical with respect to another instance of CD smell) and (iii) the *Dependency metrics* of Robert Martin [25] (Instability, Fan In, Fan Out, Efferent, and Afferent Coupling) used for the AS detection. The higher the ADI value, the higher the debt. All the details about the ADI computation can be found in our previous work on this index [4].

We ran Arcan on the commits of each considered project and organized the results in a dataset, where each observation corresponds to a single commit of a single project. The columns of the dataset store the data about 1) the project the commit belongs to 2) the number of AS detected in the commit (one column for each type) and 3) the value of the ADI of the commit. The dataset and the analysis script are available in the replication package².

3.3. Analysis

In order to answer our research questions, we conducted two kinds of analysis on the dataset (number of AS and ADI). First, we extracted a **set of statistical metrics** (mean, standard deviation, minimum value, maximum value) for each project, to ease the interpretation of the Arcan analysis results. All metrics are evaluated with respect to the analysed time period, i.e., the data extracted from the considered commits. In this way, we can compare the statistics of the different projects, even if their ATD was evaluated on time periods of different length.

We also conducted **trend analysis** to understand how ADI and AS evolve overtime. We exploited the *Mann-Kendall test*, which is a non-parametric test able to assess if there is a monotonic upward or downward trend of the variable of interest over time. In our case, given the number of AS and the ADI value for each commit, the test is able to compare the values across history (i.e., the commits ordered by time of creation) and determine whether, along time, the number of AS and ADI increases/decreases or does not show a trend. If a trend is present, it can be the first clue that the presence of AS and ADI has a relationship with other kinds of variable, i.e., the maturity of the project, the seniority of the developers, the development practices adopted by the developers and so on.

Notice that this test can be used to find trends for as few as four samples. In our case, one sample corresponds to one commit. However, with only a few analysed samples, as in the

²<https://gitlab.com/essere.lab/public/mas-atd-evaluation>

case of Jade (only 6 versions), the test has a high probability of not finding a trend when one would be present if more commits were provided. Hence, we report also the results of Jade trend analysis, but knowing that they could be less relevant with respect to the other analysed projects.

Finally, we conducted a manual validation of the results of the tests. In particular, we collected the commit comments attached to Github and the available release notes. This was useful to offer an interpretation of the results of the single projects and to acquire information useful to compare the different projects among them.

4. Results

In this section, we report the results of our analysis and also the answers to our research questions. Table 2 reports results of the distribution analysis conducted on the four projects. The statistics are evaluated on the number of AS, also divided by AS type (CD, HL, UD), and on ADI, measured for each commit during the considered time period. The project with the highest mean number of AS is Jade (≈ 879), and it has also the highest mean value of ADI (≈ 38).

On the other hand, Netlogo has the lowest AS and ADI mean values. Notice that it is reasonable to have a non-zero number of AS in large projects as the considered ones. We analysed many Open Source Java projects in past works, indeed the ADI value is tuned with a reference dataset of past analysed projects. However, to be able to define how much is a “good” amount of AS in MAS platform is not a trivial task, because the answer is largely bounded to the development context (e.g., developers, developers skills, MAS platforms peculiarity). That is why in this study we mainly focused on the evolution of ADI and in grasping some insights about why they appear/disappear.

We can provide the answer to the first RQ:

RQ1: Which is the most present type of AS in MultiAgents Systems platforms? The most present type of AS (on average) is CD. The less present AS is HL.

We also ran the Mann-Kendall tests to analyse the trend of the same variables (CD, HL, UD, AS and ADI). Table 3 reports the results only of the significant cases, i.e., with p-value < 0.05 . The table also indicates whether the *trend* is increasing (+) or decreasing (-).

We now put in relation the results of the two analysis and provide a brief discussion of the architectural debt of each project. In particular, we manually checked the commit comments of each project, with a focus on the commits which presented large drops of the ADI value (points of interest). Our aim was to find a relationship between the change in the value of ADI and the content of the commit under analysis, starting from the description reported in the commit comment by the developers. For instance, if a sudden decrease in the ADI is backed by a comment stating that a major refactoring was applied in the commit, then the Arcan result is validated and we obtain an insight about practices for the removal of ATD.

Figure 1 depicts the ADI trend (y-axis) of the projects, computed for each commit (x-axis). Table 4 reports the main points of interest in the projects commit history, identified by the *Date* of the commit, the *Commit hash*, the *ADI* value and the interesting *Characteristics* of the commit. The table does not report results concerning Jade because we conducted a different

Table 2

Distribution analysis results

Metric	Jade	Jadex	Jason	Netlogo
CD mean	846.83	123.38	61.95	9.06
CD std.dev	7.96	55.78	13.22	4.31
CD min	837	1	48	1
CD max	856	193	91	16
HL mean	5	1.62	3.34	1.00
HL std.dev	0	0.49	0.58	NA
HL min	5	1	2	1
HL max	5	2	4	1
UD mean	28	67.96	8.55	1.79
UD std.dev	1.67	28.19	2.36	0.43
UD min	27	1	7	1
UD max	31	103	15	2
AS mean	879.83	192.12	73.84	7.2
AS std.dev	7.22	83.32	13.71	6.09
AS min	869	2	59	1
AS max	888	289	110	18
ADI mean	38.33	10.80	23.45	3.96
ADI std. Dev	1.97	5.95	3.06	3.52
ADI min	35	3	19	0
ADI max	41	23	30	11

Table 3

Mann - Kendall test results

Project	P-value	Variable	Trend
Jade	0.019	ADI	-
Jadex	0.000	ADI	+
Jadex	0.000	AS	+
Jadex	0.000	CD	+
Jadex	0.043	HL	+
Jadex	0.043	UD	+
Jason	0.003	AS	+
Jason	0.000	CD	+
Netlogo	0.000	ADI	-
Netlogo	0.000	AS	-

kind of analysis on it. Given that Jade is not hosted on Github, we could not analyse the commit comments, however we manually checked its changelogs.

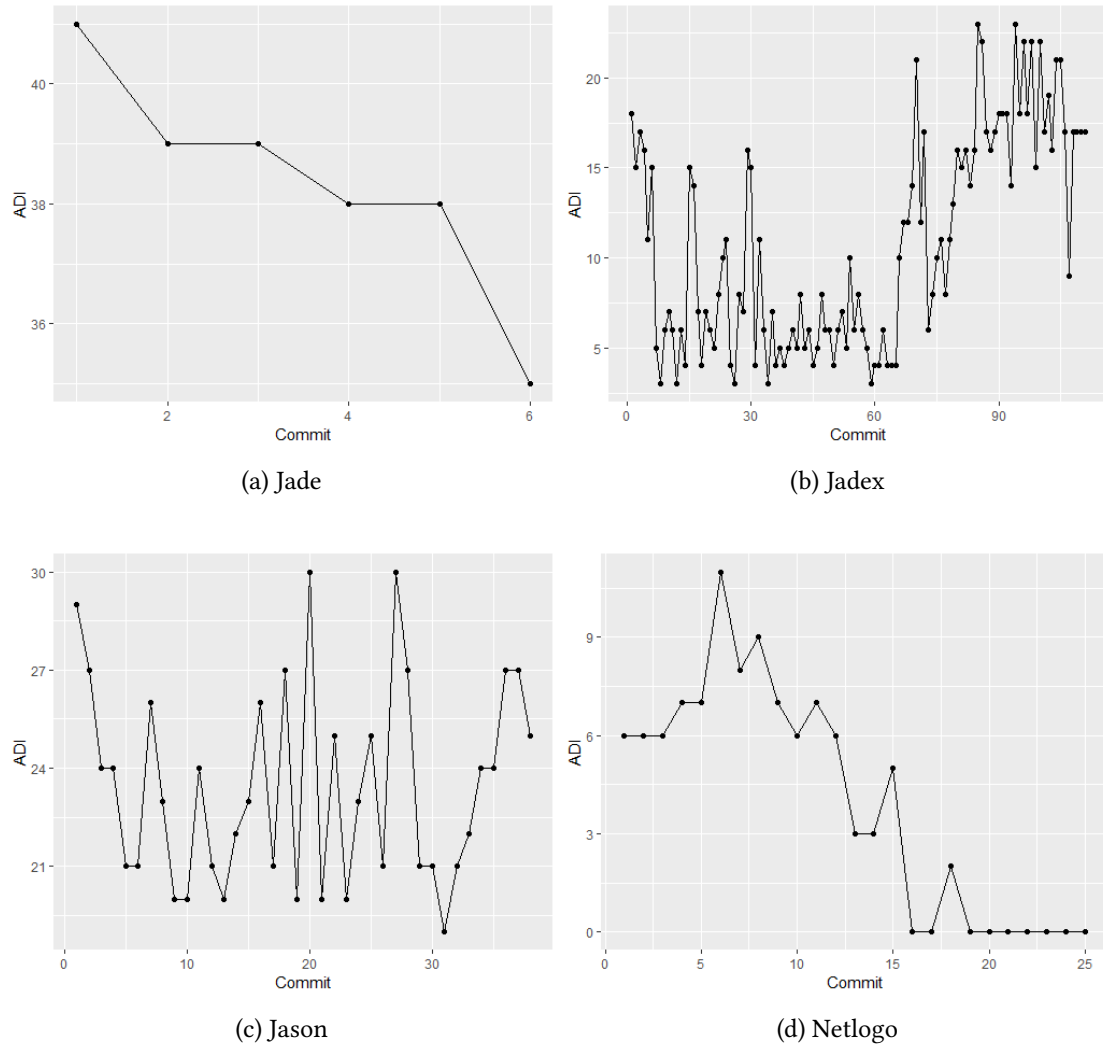


Figure 1: Evolution of ADI value of the 4 projects

4.1. Jade

As underlined before, the scarce number of analysed versions may have hindered the trend analysis results. However, the Mann-Kendall test gave an output for the ADI variable. In particular, the ADI trend is decreasing, but not dramatically. The detected ADI value ranges from 35 (last analysed version, 4.5.0) to 41 (first analysed version, 4.3.0). Given the few versions, we were able to manually analyse the changelog³ of all of them. We checked for key-terms, namely *Improvement(s)* and *Fix(es)*. We noticed that each version is characterised by many fixes, with version 4.4.0 having the greatest number of changelog comments addressing them (8). Concerning improvements, we identified few of them (approximately one per version), with

³<https://jade.tilab.com/doc/ChangeLog>

Table 4

List of ADI points of interest

Project	Date	Commit hash	ADI	Characteristics
Jadex	08/06/2009	09681d4371f53a1822de0a16c5b86e8349ea43c1	3	Min ADI value
Jadex	08/10/2009	5d266d02e4e9d90bde2dd942a7aff456eeca1aa4	3	Min ADI value
Jadex	14/12/2010	81be90b42d44a135e74d8a00213406951b78acaa	3	Min ADI value
Jadex	15/08/2011	a5b92b3f35a33222cd501bb37a2200e67d24187a	3	Min ADI value
Jadex	09/09/2014	fc28f548505583bfb2f1adcb1d3e368ec81aafd0	21	ADI drop, preceded by fixes and introduction of new data structure
Jadex	13/11/2017	6292d0cc21c24fa16627725e1bd0bfab52531222	9	ADI drop, preceded by fixes
Jason	20/04/2021	680921bbe8ff0247427d22e57ea3e36497143cd5	25	ADI drop, preceded by the implementation of a new test framework
Netlogo	07/02/2012	15daf0d82f11acc3a66ac9ca369fccf4efe77776	8	ADI drop
Netlogo	08/05/2012	5c5f707059b9a5c6546702cd2092b58e971b2632	6	ADI drop
Netlogo	17/05/2013	00ab7fae6c16c7a9b7e6b928080b15e0cd532e29	3	ADI drop
Netlogo	17/06/2013	82673cd6eda0f19b1dd533bd0e3a629ea24f23e6	3	ADI drop
Netlogo	31/01/2014	05710fe041397fd70286bc2347343993dd4f5563	0	ADI drop, corresponding to pull-request
Netlogo	13/03/2014	9056a8d98b69a2a984580d2cafceffc50685acb6	0	ADI drop, corresponding to pull-request
Netlogo	15/05/2014	c6a5902697212fb7adc14ae8d1e8a3e428e4dae8	2	ADI drop, corresponding to the addition of a new Scala submodule
Netlogo	04/09/2014	895609613fc1b5f592ff9eb84dcc5767f40ee7ec	0	ADI drop, corresponding to pull-request

most of them referring to enhancements to security. However, version 4.5.0 reports a comment about “Improved code style and logging”. A clean code style can improve maintainability, and this could be reason behind the ADI value of this version, the lowest detected.

4.2. Jadex

The ADI trend is increasing. The same happens for all the other variables (number of AS, CD, HL and UD). Indeed, Jadex is the project with the highest mean number of AS.

We manually analysed the points in time where ADI reached its lowest values, with the aim to understand whether interesting practices to manage architectural debt could emerge. In particular, we analysed the five commits corresponding to the lowest values of ADI, equals to 3 (see Table 4). Unfortunately, there are no messages or comments associated to those specific commits. The only interesting aspect is that all the five commits were created by the same two authors. We also analyse the period of time between and 09/09/2014 comments report multiple time the word “fix” and also the adoption of a dedicated info structure for Non-Functional

properties (NFPropertyInfo class). which are Non-functional property annotation.

Another point of interest in the Jadex history is on date 13/11/2017, when ADI drops from value 17 to 9. We checked the commit comments between the two points, corresponding to the changes made in a month, and all of them concern fixes. Some examples: “**Fix** proxy factory class loader issue and component spec as class.”; “**Fixed** most test failures caused by “config cleanup” commits”; “**Fix** component/bootstrap factory stored as string and as class”.

4.3. Jason

This projects ADI does not show any trend. However, its number of AS and CD has an increasing trend. We manually analysed a sampled period, which comprises the commits between 17/08/2020 and 20/04/2021. First we analysed the period from the high peak (ADI=30) to the lowest point (ADI=19). From the commit comments and the changelog of the nearest release, it appears that the most meaningful development was the implementation of a new tests framework. Even if it is affected by less AS with respect to Jadex, Jason is the project with the highest average ADI value. This means that compared to Jadex, its AS are more critical (have highest severity [4]).

4.4. NetLogo

NetLogo ADI trend is the only one decreasing. At the same time, the number of AS has a decreasing trend: we can deduce that the decrease of the value of ADI is not due to the decreased severity of the smells, but only due to the decrease of the total number of smells. In general, Netlogo is the projects less affected by architectural smells and with the lowest values of ADI (see Table 2).

We manually analysed the commit history of this project, in particular we focus on the points where ADI decreases (see Table 4). Most of the associated commit messages indicate improvements: “**Minor improvement** to Client Perspective Example.”; “**Mostly-irrelevant correction** to a HubNet method’s Scala style”. However, there are no signs of big, structural changes which could explain the significant drops of the ADI value, apart from the presence of three pull-requests, corresponding to $ADI = 0$ and the introduction of a new Scala submodule providing network analysis tools for use in NetLogo (commit message: “Add new network extension submodule!”).

RQ2: What can we observe according to architectural debt of the considered MultiAgents Systems platforms? All the analysed projects present architectural debt along their development history, but with different trends. Jadex has an increasing ADI, while Jade and Netlogo show a descreasing trend, with Netlogo having the last commits with zero debt. Jason did not present any trend.

5. Conclusions

We exploited our tool Arcan to analyse four Open-Source MultiAgent Systems (MAS) development platforms and we evaluated their Architectural Technical Debt (ATD). We investigated the

outcome of the tool by manually analysing the commit comments available on Github, for three of the four projects, and the changelogs for one of them. From our analysis, we acknowledged that the considered MAS platforms are affected by architectural debt, in particular Jade is the most affected, while Netlogo is the less affected, with a decreasing ADI trend.

From the manual analysis, we could not find clear indication of practices to manage architectural debt. However, for all the projects, in the points in time where ADI reaches its minimum, the comments refer to “Bug fixing”, “Improvements” or pull requests. This could mean that architectural debt, usually considered only at architectural level, has also a relationship with issues at code level, such as bugs.

Our findings suggest us possible future works. As just outlined, studying the correlation between MAS platform architectural debt and bugs could lead to the conclusion that code level bugs have an impact on the accumulation/decrease of ATD. Moreover, the validation of the ATD values found in the projects could be refined by testing the correlation with issues coming from issue trackers (e.g. Jira⁴). In this way, we could further investigate whether “Improvements” (which is usually recognized as a category of issues) do have a relationship with the decrease of ATD. Another interesting study could investigate the relationship between ATD and MAS platforms’ performance, since a link between performance and design decisions has already been proven (see Section 2).

Another next natural step will be to apply this kind of analysis to real MASs developed with these platforms, or to enlarge our analysis to the many add-ons of these four platforms (for example WSIG and OntologyBeanGenerator for Jade [26, 27, 28]), to study if we can identify some common ATDs for MASs or further problems in MAS development platforms. We have a long experience in developing large and real MASs ([29, 30, 31]), and we would search for other concrete examples of large MASs to be analyzed from this architectural point of view.

References

- [1] Z. Li, P. Liang, P. Avgeriou, Architectural debt management in value-oriented architecting, in: *Economics-Driven Software Architecture*, Elsevier, 2014, pp. 183–204.
- [2] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, I. Gorton, Measure it? manage it? ignore it? software practitioners and technical debt, in: *Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015*, ACM, New York, NY, USA, 2015, pp. 50–60.
- [3] W. Cunningham, The wycash portfolio management system, *ACM SIGPLAN OOPS Messenger* 4 (1992) 29–30.
- [4] R. Roveda, F. Arcelli Fontana, I. Pigazzini, M. Zanoni, Towards an architectural debt index, in: *Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Technical Debt track, IEEE, Prague, Czech Republic, 2018.
- [5] G. Digkas, M. Lungu, A. Chatzigeorgiou, P. Avgeriou, The evolution of technical debt in the apache ecosystem, in: *European Conference on Software Architecture*, Springer, 2017, pp. 51–66.

⁴<https://www.atlassian.com/software/jira>

- [6] S. S. de Toledo, A. Martini, A. Przybyszewska, D. I. Sjøberg, Architectural technical debt in microservices: a case study in a large company, in: 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE, 2019, pp. 78–87.
- [7] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison, Hidden technical debt in machine learning systems, *Advances in neural information processing systems* 28 (2015) 2503–2511.
- [8] I. Pigazzini, F. Arcelli Fontana, Evaluating the architectural debt of iot projects, in: 3rd International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT 2021), 2021.
- [9] R. C. Cardoso, A. Ferrando, D. Briola, C. Menghi, T. Ahlbrecht, Agents and robots for reliable engineered autonomy: A perspective from the organisers of AREA 2020, *J. Sens. Actuator Networks* 10 (2021) 33. URL: <https://doi.org/10.3390/jsan10020033>. doi:10.3390/jsan10020033.
- [10] V. Mascardi, D. Weyns, A. Ricci, Engineering multi-agent systems: State of affairs and the road ahead, *ACM SIGSOFT Softw. Eng. Notes* 44 (2019) 18–28. URL: <https://doi.org/10.1145/3310013.3310035>. doi:10.1145/3310013.3310035.
- [11] F. Zambonelli, A. Omicini, Challenges and research directions in agent-oriented software engineering, *Auton. Agents Multi Agent Syst.* 9 (2004) 253–283. URL: <https://doi.org/10.1023/B:AGNT.0000038028.66672.1e>. doi:10.1023/B:AGNT.0000038028.66672.1e.
- [12] R. Calegari, G. Ciatto, V. Mascardi, A. Omicini, Logic-Based Technologies for Multi-Agent Systems: Summary of a Systematic Literature Review, *International Foundation for Autonomous Agents and Multiagent Systems*, Richland, SC, 2021, p. 1721–1723.
- [13] F. Arcelli Fontana, I. Pigazzini, R. Roveda, D. A. Tamburri, M. Zanoni, E. D. Nitto, Arcan: A tool for architectural smells detection, in: *Int'l Conf. Software Architecture (ICSA 2017) Workshops*, Gothenburg, Sweden, 2017, pp. 282–285. doi:10.1109/ICSAW.2017.16.
- [14] O. F. Rana, K. Stout, What is scalability in multi-agent systems?, in: *Proceedings of the fourth international conference on Autonomous agents*, 2000, pp. 56–63.
- [15] R. Endsuleit, J. Calmet, et al., A security analysis on jade (-s) v. 3.2, in: *Proceedings of NORDSEC*, Citeseer, 2005, pp. 20–28.
- [16] L. Mulet, J. M. Such, J. M. Alberola, Performance evaluation of open-source multiagent platforms, in: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 2006, pp. 1107–1109.
- [17] J. M. Alberola, J. M. Such, A. Garcia-Fornes, A. Espinosa, V. Botti, A performance evaluation of three multiagent platforms, *Artificial Intelligence Review* 34 (2010) 145–176.
- [18] F. A. Fontana, F. Locatelli, I. Pigazzini, P. Mereghetti, An architectural smell evaluation in an industrial context, in: *The Fifteenth International Conference on Software Engineering Advances, ICSEA 2020*, 18-22 October 2020, Porto, Portugal, 2020.
- [19] F. L. Bellifemine, G. Caire, D. Greenwood, *Developing Multi-Agent Systems with JADE*, Wiley, 2007.
- [20] L. Braubach, W. Lamersdorf, A. Pokahr, Jadex: implementing a bdi-infrastructure for jade agents, *EXP In Search of Innovation (Special Issue on JADE)* 3 (2003).
- [21] R. H. Bordini, J. F. Hübner, M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*, John Wiley & Sons, Inc., USA, 2007.

- [22] U. Wilensky, NetLogo, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999. URL: <http://ccl.northwestern.edu/netlogo/>.
- [23] N. Nagappan, T. Ball, Using software dependencies and churn metrics to predict field failures: An empirical case study, First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), 2007, pp. 364–373.
- [24] E. Kouroshfar, M. Mirakhorli, H. Bagheri, L. Xiao, S. Malek, Y. Cai, A study on the role of software architecture in the evolution and quality of software, volume 2015-Augus, IEEE International Working Conference on Mining Software Repositories, IEEE Computer Society, 2015, pp. 246–257. doi:10.1109/MSR.2015.30.
- [25] R. C. Martin, Object oriented design quality metrics: An analysis of dependencies, ROAD 2 (1995).
- [26] D. Briola, V. Mascardi, M. Gioseffi, Ontologybeangenerator 5.0: Extending ontology concepts with methods and exceptions, in: M. Cossentino, L. Sabatucci, V. Seidita (Eds.), Proceedings of the 19th Workshop "From Objects to Agents", Palermo, Italy, June 28-29, 2018, volume 2215 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 116–123. URL: http://ceur-ws.org/Vol-2215/paper_19.pdf.
- [27] C. van Aart, R. Pels, G. Caire, F. Bergenti, Creating and using ontologies in agent communication, in: Proc. of OAS, 2002.
- [28] M. Tomaiuolo, F. Bergenti, A. Poggi, P. Turci, Owlbeans - from ontologies to java classes, in: M. Baldoni, F. D. Paoli, A. Martelli, A. Omicini (Eds.), WOA 2004: Dagli Oggetti agli Agenti. 5th AI*IA/TABOO Joint Workshop "From Objects to Agents": Complex Systems and Rational Agents, 30 November - 1 December 2004, Torino, Italy, Pitagora Editrice Bologna, 2004, pp. 116–125.
- [29] D. Briola, V. Deufemia, V. Mascardi, L. Paolino, Agent-oriented and ontology-driven digital libraries: the indianamas experience, *Softw. Pract. Exp.* 47 (2017) 1773–1799. URL: <https://doi.org/10.1002/spe.2494>. doi:10.1002/spe.2494.
- [30] M. Leotta, S. Beux, V. Mascardi, D. Briola, My mood, a multimedia and multilingual ontology driven MAS: design and first experiments in the sentiment analysis domain, in: C. Bosco, E. Cambria, R. Damiano, V. Patti, P. Rosso (Eds.), Proceedings of the 2nd International Workshop on Emotion and Sentiment in Social and Expressive Media: Opportunities and Challenges for Emotion-aware Multiagent Systems co-located with 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Istanbul, Turkey, May 5, 2015, volume 1351 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 51–66. URL: <http://ceur-ws.org/Vol-1351/paper4.pdf>.
- [31] V. Mascardi, D. Briola, D. Ancona, On the expressiveness of attribute global types: The formalization of a real multiagent system protocol, in: M. Baldoni, C. Baroglio, G. Boella, R. Micalizio (Eds.), AI*IA 2013: Advances in Artificial Intelligence - XIIIth International Conference of the Italian Association for Artificial Intelligence, Turin, Italy, December 4-6, 2013. Proceedings, volume 8249 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 300–311. URL: https://doi.org/10.1007/978-3-319-03524-6_26. doi:10.1007/978-3-319-03524-6_26.