STUDIA INFORMATICA 2014
Volume 35 Number 2 (116)

Artur OPALIŃSKI
Politechnika Gdańska, Wydział Elektrotechniki i Automatyki

CONSOLIDATING LOGS IN VARYING FORMATS INTO A STRUCTURED REPOSITORY

Abstract. During multi-agent systems development, testing and research, possibly voluminous logs are created at distributed agent locations. These logs describe individual agents' behavior and state changes. To get the complete picture of the multi-agent system working, it is necessary to consolidate these logs and to store them in a format appropriate for diverse analyzes. Specific issues arise during consolidation if the log formats are not uniform. This paper presents an approach to consolidate logs in various formats into a single repository suitable for further analyzes.

Keywords: data consolidation, data extraction, meta-data management

KONSOLIDACJA LOGÓW W ZMIENNYCH FORMATACH DO STRUKTURALNEGO REPOZYTORIUM

Streszczenie. Podczas rozwijania, testowania i badania systemów wieloagentowych tworzone są obszerne logi w rozproszonych lokalizacjach, w których działają agenty. Logi opisują zachowania poszczególnych agentów oraz zmiany ich stanów. Żeby uzyskać kompletny obraz działania systemu wieloagentowego, konieczna jest konsolidacja tych logów do postaci przydatnej dla różnorodnych analiz. Niejednolity format logów różnych agentów tworzy specyficzne wyzwania. Artykuł opisuje podejście do konsolidacji logów w zmiennych formatach do postaci jednego repozytorium, przydatnego do prowadzenia dalszych analiz.

Słowa kluczowe: konsolidacja danych, ekstrakcja danych, zarządzanie meta-danymi

1. Introduction

The problem considered in this paper is to consolidate multiple voluminous logs in various formats into a single repository, suitable for further analysis. This problem is especially valid for multi-agent systems. Multi-agent systems constitute a fast-growing domain of software agent-oriented technology, which deals with knowledge management and autonomous decision taking. During research of multi-agent systems, individual agents are producing logs of their activities. These logs can be voluminous, depending on the number and activity of agents at play, as the system is usually under development in the research phase, and different kinds of decisions, state and state changes, debugging, and other information gets logged. Discovering sequences of agents decisions, or agents communication, or other related events is crucial to track and match causes and their effects, and to understand and explain system behavior. Particularly debugging and understanding multi-agent systems is a challenging task due to the number of agents involved and the complex communication patterns that they may exhibit. To research the features and behavior of a multi-agent system as a whole, a unified analysis of the logs is valuable. For this, it is necessary to select the useful content from separate agents' logs and to consolidate it into a single repository. Unfortunately the logs are usually produced at distributed places. Agents in the system may not be homogeneous, so the log formats and logging mechanisms can differ. Also log formats in development are usually not uniform even for a single agent's run. Therefore it is not obvious how to consolidate these multiple logs while limiting human involvement.

The goal of this paper is to describe a centralized, relational log database with the capability of importing logs despite their varying formats and unspecified field types.

The thesis is that by working in concert on the agent and on the repository side it is possible to recognize log format and assume a sufficient data type representation to store multi-agent system text logs in relational-database format suitable for later queries. Querying the database is out of the scope of this paper.

2. State of the Art

Considerations of load of information on humans in relation to software agents has a relatively long history, starting with [1] and [2]. Current research in mutli-agent systems which relies heavily on logging includes [3], which documents research on execution patterns in multi-agent systems. [4] and [5] attempt to semantically interpret distributed agents system logs, to reconstruct a detailed view of agents' decisions during incident management in indus-

trial facilities. [6] exploits logs to improve multi-agent system stability and robustness. Authors describe logging-based infrastructure methodology for analysis of agent problem-handling. Analyzing past agents behavior allows to recover from errors and thus to repair already-completed as well as current goals. The infrastructure is based on domain-specific knowledge related to changes in goal status and semantic compensations. There are similar solutions to increase multi-agent systems reliability [7] which also require logs consolidation for analysis of components or transaction failures. Monitoring remains an obvious application for log consolidation, too [8]. [9], [10] and [11] deal with extracting information from event logs in multi-agent and other systems. The audit trails of a system workflow is used to discover models describing processes. AgentScope tool described in [12] implements measurement and analysis interface for developing multi-agent systems. It uses a logbook abstraction to collect data and a backend adaptor implementation of a Logger interface to route the individual, distributed logbooks to a central logbook location. Processing the logs is crucial for measurements and analysis in this work.

To take advantage of the aforementioned AgentScope tool, it is necessary to use its set of programming interfaces. Therefore it is not easily available for creating agents in different programming languages, nor in multi-agent systems where source code is not available for all agents. In contrast, the solution presented in this paper is agnostic to the technologies and languages in which agents are built.

There is a recent attempt to consolidate logging by employing a single log agent [13], but this does not solve the development phase challenges, where every agent has to do and does some logging. Using a dedicated log agent introduces some dependencies, which may not be met at early stages of development: communication path must exist, agents must be identifiable and perhaps registered for communication, agents must implement specific protocols for log expediting, etc. Nevertheless even at the early stages agents may need be run for debugging or testing.

[14] proposes visualization of agents communication and tasks. The visualization can be performed offline, i.e. based on logs, but is aimed at tracking only predefined events, e.g. agent's start, communication, movement, action, end. The events can be defined at will, but requires logging in XML format, defined in a Document Type Definition file, this again creates an additional work when applied in the development phase.

Logs happen to be stored in databases, either in relational databases [15] or in so called NoSQL databases [16, 17], like Apache Hadoop [18], Apache Cassandra [19], Hive [20], or Hbase [21]. Log entries are sometimes stored without parsing, e.g. as BLOBS [15], and then the problem of preparing data for analysis remains open. Other solutions bear the assumption that dedicated parsing code has to be created during database setup, like writing [22] func-

tions to parse each type of input data for aforementioned Hadoop. The latter is acceptable in production or otherwise stable environments, but it is awkward during development and research phase, when the format and logical contents of the logs changes frequently. Yet other solutions force to use dedicated logging API [23]. This excludes direct of agents for which source code is not available, without introducing intermediate translation layers. There are multi-agent systems possible, where agents are not homogeneous, e.g. when agents are developed for very different platforms, including resource-constrained ones (e.g. microcontroller-based sensors) where an API library is not an option.

All the existing approaches assume that format of log entries remain constant over time. This is generally not true when developing software to research resulting multi-agent systems: multiple parts of the software usually outputs different state and action information, which may influence not only future actions and state changes of the same agent, but also the state changes of and actions taken by other agents.

3. Solution

The basic assumption in the presented solution is that creating logs should not attract much attention when developing and researching multi-agent systems. The usual text logs are therefore used. It is also assumed that numerous changes to the multi-agent system should not entail changes in the logging subsystem. Typical changes in the multi-agent system, related to logging, include:

- subset of internal state variables reported,
- order in which variables from the selected subset are reported,
- type of the logged values.

Hence it is not evident which data is actually stored in a log field for such temporary, ad-hoc changes. Additional information on the side of the agent is therefore needed. The minimal change on the side of the agent code is to ensure that:

- log lines contain fields which are delimited by separators. This is necessary anyway when coding logging, because usually many values describe agent's state and need to be logged together.
- when log format changes, e.g. due to following a different execution path of agent's code – a description of the currently used log format is provided, in the form of a header line. A header line must start with a prefix. The header line should contain log field names, delimited by separators. Header lines must be coded anyway in more complex logging; a feasible solution is then to precede each log line with a header line. Starting with the

header line, the new log format is used for parsing log lines, up to the next header line or to the end of input.

Agents can be heterogeneous, but their exact capabilities do not play any role as long as they allow to save their logs to files, and use delimited fields in log lines. To provide for the above requirements:

- if agent source is available, agent's logging behavior must be modified by adding a header line at least at the points when log output changes; alternatively, corresponding header line may precede each log line.
- if agent's source is unavailable, its log line format must be stable. If the agent does not provide a preceding header line by itself, one must be fed from a separate, manually created file.

The complete assumptions of the solution proposed in this paper are:

- Multi-agent systems research requires many changes both in agent code, as well as in their environment.
- During runs, each agent generates its own log; agents may all run locally, or may be distributed.
- Logs contain numerous text lines, with many simple value fields delimited by character sequences predefined for the consolidation process. These fields reflect agent state or decisions.
- The fields of the log line can be integer numbers, floating point numbers, or strings.
- The type of fields is not specified and may change between runs. This accommodates for the fact that different aspects of agent state or working are important during development and research in separate runs. The type of fields must be automatically detected during consolidation. As a result of the auto-detection, previous assumptions about value types may be invalidated.
- Log format is unstable during development. The number of fields and their order in log lines may change during run, depending on the code path being executed.
- The names of the fields in the log line must be given in a preceding header line.

The logs are read line per line (Fig. 1), and log lines get parsed according to description provided by header lines. Comment lines are skipped.

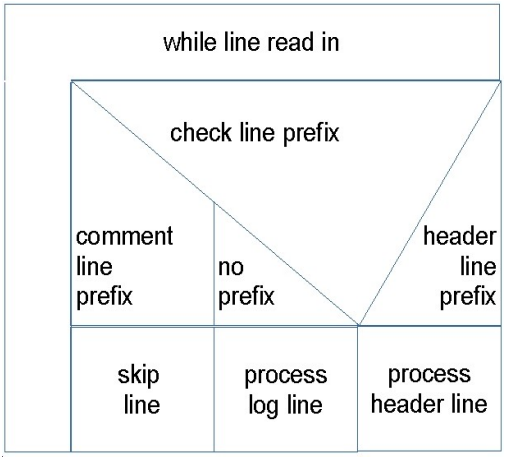

Fig. 1. High-level Nassi–Shneiderman diagram of log processing algorithm
Rys. 1. Ogólny diagram Nassi–Shneidermana algorytmu przetwarzania logów

It has been decided to store the values of the log line fields in relational database table named log_table (see Fig. 2). This table stores log fields as separate attributes. Therefore the complete structure of this table is not known in advance: the number of attributes related to the number of log fields is initially unknown. As header lines with previously unseen fields are encountered during consolidation, this table is extended by adding the appropriate number of attributes.

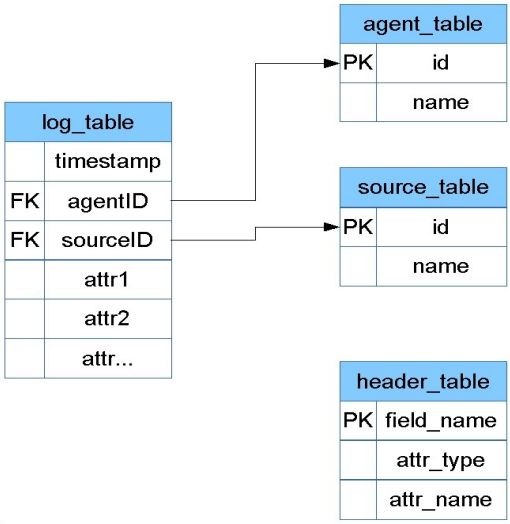

Fig. 2. Database diagram
Rys. 2. Diagram bazy danych

Names of fields are only presented in header lines, but are necessary for describing log line fields during analysis of the consolidated logs. Thus at encountering a header line (Fig. 3), the newly-generated attribute name of the log_table is recorded in the header_table, alongside the corresponding name of the field given in the header line. The attribute header_table.field_name must be unique to identify the same log field even when the order of fields changes. This also allows to query for a given log field across all the agents. Of course,

the validity of such queries depends on ensuring unique log field names in use by all agent types, for the fields with the same meaning.

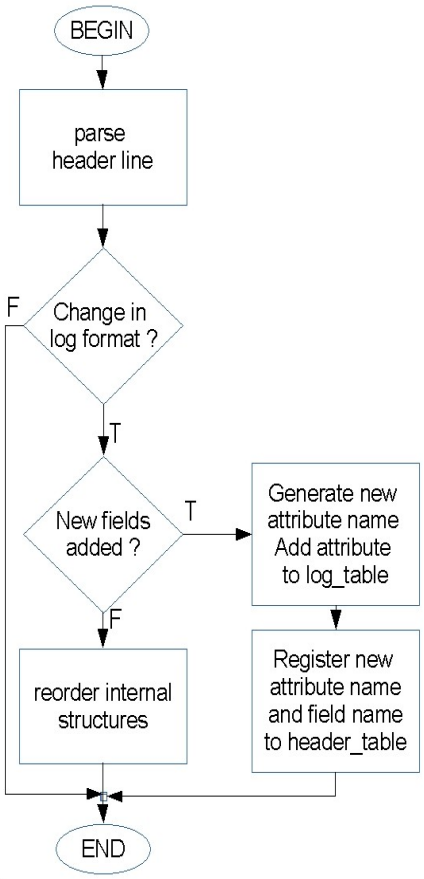

Fig. 3. Control flow diagram for header line processing
Rys. 3. Diagram sterowania podczas przetwarzania linii nagłówkowych

Types of the field values are unknown in advance, and are not specified by the header lines. As only three types of field values are permitted, the type of the field values is auto-detected. By default any field value is initially treated as integer number. Promotions to real number or to character string are possible. No other field value types are taken into account in the processed log lines. Only promotion to higher-ranked type is possible. The automatic check is performed for every value in every log line at the time it is read in, in the form of character string.

When a field value is detected to not comply with integer number format, it is treated as floating-point number. When a floating-point field value is detected to not comply with floating-point number format, it is treated as character string and no further checks are conducted with this field. It is assume that the character representation N of integer numbers is a subset of character representation R of real numbers, which is in turn a subset of character string S, i.e.:

$$N \subseteq R \subseteq S \tag{1}$$

The currently assumed field value type is recorded in header_table. When changing the assumption about value type of a log field, the type of the corresponding attribute of the log_table must be changed. This is time-consuming and consists of three tasks:

- header_table must be updated with new type,
- this attribute's definition in the log_table must be altered,
- previously stored values of this attribute must be converted to the new type.

Agent's identifier and source of the log get recorded in the database, in the agent_table and source_table, respectively. For the sake of later analysis, each log line should contains a timestamp allowing to order events by timeline. The timestamp is recorded in the log_table.

The usage scenario is as follows:

- Logs from agents' run are saved to separate text files during runs, and used offline.
- After agents' run, their text log files are one by one consolidated into a database (this is currently automated by a shell script).
- It is up to the user to select which logs to consolidate. This is the responsibility of the user to only analyze logs from comparable runs, and to avoid importing logs which contain unrelated information, especially unrelated fields in log files, which would otherwise unnecessarily clog the database. Generally separate databases should be created per each run.

4. Verification

During verification SQLite [24] has been employed to provide an embedded relational database. SQLite is a public domain software library that implements a self-contained, serverless, transactional SQL database engine. The library has many language bindings, either native or through third-party wrappers. The native C language bindings have been used to code a proof-of-concept program, which implements the algorithms and database structure as depicted in Fig. 2.

The proof-of-concept program [25] allows to configure comment line and header line prefixes, field separators, and case sensitivity, as well as the name of the output database to create or update.

The proof-of-concept program and verification results confirmed that by working in concert on the agent and on the repository side it is possible to recognize log format and assume a sufficient data type representation to store multi-agent system text logs in relational-database format suitable for later queries.

Table 1

Verification results. IF: total number of input files, FO: number of times field order has been reversed, FN: change in number of fields, FT: number of fields requiring promotion

Case	Parameters	Initial lines	Initial fields	Run time [s]	Fields per sec
A	IF=1	10000	100	27.917	35820
A	IF=1	10000	1000	401.975	24877
A	IF=1	100000	100	279.889	35728
A	IF=5	100000	100	279.879	35730
A	IF=10	100000	100	279.885	35792
A	IF=1	100000	1000	3982.545	25110
B	FO=1	100000	100	279.874	35730
B	FO=99999	100000	100	281.004	35578
C	FN=-1	100000	100	279.881	35729
C	FN=-99	100000	100	231.519	43193
C	FN=1	100000	100	279.919	35725
C	FN=99	100000	100	389.791	25655
D	FT=1	100000	100	283.583	35263
D	FT=50	100000	100	384.348	26018
D	FT=100	100000	100	489.791	10417

The results from parametrized verification cases are presented in table 1. Functionality in each case has been proved by using simple SQL SELECT queries. Additionally time of the runs was measured with millisecond accuracy, using high-resolution time provider in Microsoft Windows [26]. An average run time from tree runs is given in the table. The cases were as follows:

- Case A: loading to the database log files with stable formats. Each input file contains the same number of lines. The total number of input files is *IF*.
- Case B: loading to the database a log file with varying format. Field number was constant. Field order changed *FO* times.
- Case C: loading to the database a log file with varying format. Change in number of fields by *FN*. Positive *FN* values mean stepwise increase by that number of fields. Negative *FN* values mean stepwise decrease by that number of fields.
- Case D: loading to the database a log file with varying format. Promotion of type of *FT* fields from integer to character string in the last log line, i.e. when all lines except the last one have already been loaded.

Verification case A demonstrates that the consolidation time does not depend on the number of input files, which are accessed sequentially. For constant log format, consolidation time depends linearly on the number of log lines and the number of fields in each log line.

Verification case B shows that changes in field order do not influence consolidation time. This is due to the fact that mapping between log field names and database attributes is stored

in memory and reordered according to the last header processed. Changes in field order in log file do not result in changes to the database schema.

Changing log file format in verification case C by decreasing the number of fields does not result in changes to the database schema. The processing time drops, because less fields are to be processed. When (FN=-99) average number of fields decreases to 50% of the initial value, processing time drops proportionally. Increasing the number of fields in contrast requires adding new tuples to the header_table and - more important - adding new attributes to the log table. Additionally more fields are to be processed. Thus the processing time increases significantly.

From verification case D it can be seen that the processing time increases significantly when many field value types need to be updated when processing the last column. Updating field value types requires lightweight changes in the header_table, but also converting all the values already existing in the table, which is time-consuming. Currently a new table column is added with the desired type, but the previous column is not removed when assumption about field value type changes, because deleting columns is not supported by SQLite. SQLite also does not allow to alter exiting table column definition. The previous column could be used for promotions of other attributes – but currently it is not, so it remains unused after promotion.

5. Verification

Creating the proof-of-concept program and results of verification process confirmed that by working in concert on the agent and on the repository side it is possible to recognize log format and assume a sufficient data type representation to store multi-agent system text logs in relational-database format suitable for later queries. It has been achieved with only two requirements on the agent's side. These requirements are easy to fulfill when coding logging:

- fields in the log lines must be delimited with a field separator,
- whenever log format changes, a header line must precede and explain the forthcoming log lines.

Fulfilling the above two easy assumptions allows for adapting to changes in log format, including auto-detecting and auto-correcting data types in repository.

No modification is necessary for agents with a single, stable log format. If they do not provide header line explaining log columns, the header line can be provided from a separate, manually created file.

Agents do not need to be homogeneous in regards to logging mechanisms or formats – field separator can be set individually for any log file consolidated with the database.

While initially not planned for real-time log consolidation, the performance allows to store logs at the rate of up to approximately 35.000 fields per second when log format is stable, and over 20.000 fields per second when it is not. From author's experience an agent usually produces 100 to 1000 fields per second, depending on the number of agents, agent algorithm complexity, amount of inter-agent communication and its performance, and logging verbosity level. Te achieved performance does not preclude real-time application

Future work may include performance enhancements by either utilizing database table columns freed by previously type-promoted fields, or by utilizing other database architectures, including NoSQL [17] architectures.

BIBLIOGRAPHY

1. Sheth B., Maes P.: Evolving Agents for Personalized Information Filtering. Proc. of the 9th Conference on Artificial Intelligence for Applications, IEEE Computer Society Press, 1993.
2. Maes P.: Agents that reduce work and information overload. Communications of the ACM, Vol. 37, 1994, p. 31÷40.
3. Gutiérrez C., García-Magariño I.: Extraction of Execution Patterns In Multi-Agent Systems. IEEE LA Transactions, Vol. 8, No. 3, 2010.
4. Ilie S., Scafes M., Badica C., Neidhart T., Pinchuk R.: Semantic logging in a distributed multi-agent system. Proc. of International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010, p. 265÷270.
5. Pavlin G., Kamermans M., Scafej M.: Dynamic process integration framework: Toward efficient information processing in complex distributed systems. Proc. of 3rd International Symposium on Intelligent Distributed Computing (IDC2009), Springer Verlag, Studies in Computational Intelligence, Vol. 237, 2009, p. 161÷174.
6. Unruh A., Bailey J., Ramamohanarao K.: Building More Robust Multi-Agent Systems Using a Log-based Approach. Web Intelligence and Agent Systems, Vol. 7, Issue 1, IOS Press Amsterdam, 2009, p. 65÷87.
7. Varakantham P. R., Gangwani S. K., Karlapalem K.: On Handling Component and Transaction Failures in Multi Agent Systems. ACM SIGecom Exchanges-Chains of commitment, Vol. 3, Issue 1, 2002, p. 32÷43.
8. Figueiredo J., Lau N., Pereira A.: Multi-Agent Debugging and monitoring framework. Proc. Of First IFAC Workshop on Multivehicle Systems (MVS), 2006.

9. Rozinat A., Zickler S., Veloso M., van der Aalst W. M. P., McMillen C.: Analyzing multi-agent activity logs using process mining techniques. [in:] Asama H., Kurokawa H., Ota J., Sekiyama K. (eds.): Distributed Autonomous Robotic Systems 8 (9th International Symposium, Tsukuba, Ibaraki, Japan, November 17-19, 2008), part IV, Springer, 2009, p. 251÷260.
10. Reijers H. A., Song M., Jung B.: Analysis of a Collaborative Workflow Process with Distributed Actors. Information Systems Frontiers, 2008.
11. Process Mining tool home page, http://processmining.org/ (DOA: 15.12.2013).
12. Ogston E., Brazier F.: AgentScope: Multi-Agent Systems Development in Focus. Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), 2011, p. 389÷396.
13. Bhat S., Wahid A.: Log Agent for Knowledge Management Based on Multi-Agent System. International Journal of Emerging Technology and Advanced Engineering, Vol. 2, Issue 7, 2012.
14. Ilarri S., Serrano J. L., Mena E., Trillo R.: 3D Monitoring Of Distributed Multiagent Systems. International Conference on Web Information Systems and Technologies (WEBIST), available online: http://sid.cps.unizar.es/PUBLICATIONS/POSTSCRIPTS/webist07-agents.pdf, 2007 (DOA: 15.12.2013).
15. Product documentation: WebSphere Application Server (IBM i), Version 8.5, Chapter: Storing logs in a relational database for high availability, available online: http://pic.dhe.ibm.com (DOA: 15.12.2013).
16. Marcus A.: The NoSQL Ecosystem. [in:] Brown E., Wilson G. (eds.): The Architecture of Open Source Applications. Available online: http://www.aosabook.org/en/nosql.html, (DOA:15.12.2013).
17. Kovacs K.: A technical comparison of NoSQL products. Available online: http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis, (DOA: 15.12.2013).
18. White T.: Hadoop: The Definitive Guide. 2009.
19. Capriolo E.: Cassandra High Performance Cookbook. 2011.
20. Facebook White Paper, Facebook's Petabyte Scale Data Warehouse using Hive and Hadoop, 2010, available online: http://www.sfbayacm.org/wp/wp-content/uploads/2010/ 01/sig_2010_v21.pdf, (DOA: 15.12.2013).
21. Dimiduk N., Khurana A.: HBase in Action. 2012.
22. Dean J., Ghemawat S.: MapReduce: Simplified Data Processing on Large Clusters. Proc. of Sixth Symposium on Operating System Design and Implementation (OSDI), 2004.

23. MongoDB Ecosystem, part of MongoDB Manual, Chapter Storing Log Data. Available online: http://docs.mongodb.org/ecosystem/use-cases/storing-log-data/ (DOA: 15.12.2013).
24. SQLite library project home page. Available online: http://www.sqlite.org/ (DOA: 15.12.2013).
25. The proof-of-concept program, as set of Code::Blocks projects in C. Available online: https://sites.google.com/site/flecabinet/downloads/BDAS2014.zip (DOA:15.12.2013).
26. Nilsson J.: Implement a Continuously Updating, High-Resolution Time Provider for Windows. MSDN Magazine 2004, available online: http://msdn.microsoft.com/en-us/magazine/cc163996.aspx (DOA: 15.12.2013).

Wpłynęło do Redakcji 23 grudnia 2013 r.

Omówienie

Problem, którego rozwiązanie opisano w artykule, polega na konsolidacji wielu obszernych logów w odmiennych formatach do jednego repozytorium. Takie repozytorium służy następnie do analizy logów, która pozostaje poza tematyką artykułu. Potrzebę konsolidacji rozważono szczególnie w świetle tworzenia, testowania i badania cech systemów wieloagentowych. Generują one potencjalnie obszerne logi, szczególnie gdy system składa się z wielu aktywnych agentów, a tworzenie, testowanie czy badanie wymaga zapisywana różnych rodzajów decyzji, stanów czy zmian stanu. W tak dużej ilości informacji, rozproszonych w odrębnych logach, trudne jest śledzenie sekwencji decyzji podejmowanych przez agenty, schematów komunikacji czy innych zdarzeń istotnych dla zrozumienia działania systemu.

Problematyka zalewu informacji w agentach programowych ma dość długą historię, rozpoczynającą się od [1] oraz [2]. Aktualnie prowadzone badania systemów wieloagetowych i ich logów są opisane np. w [3], gdzie udokumentowano odtwarzanie wzorców działania (*execution patterns*) w systemach wieloagentowych.

Podstawowym założeniem proponowanego rozwiązana jest to, żeby nie wnosić istotnych zmian do metod generowania logów ze względu na to, że w heterogenicznych systemach wieloagentowych niektóre platformy sprzętowe mogą mieć bardzo ograniczone zasoby, w przypadku innych zaś może nie być dostępu do kodu źródłowego. Przyjmuje się, że na etapie testowania format logów może dodatkowo zmieniać się często, a nawet może nie pozostawać stały w trakcie jednego uruchomienia. Zdefiniowano jednak pewne podstawowe informacje,

które muszą być przekazane przez agenta, żeby umożliwić łączenie logów o różnych i zmiennych formatach.

- Linie logów muszą składać się z pól rozdzielonych separatorami. Jest to typowe, ponieważ stan agenta jest często opisany wieloma zmiennymi, których wartości muszą być raportowane jednocześnie.
- Przy każdej zmianie formatu logowania, np. w wyniku wykonania innej ścieżki w kodzie agenta, agent musi dostarczyć opis aktualnego formatu w postaci linii nagłówkowej, rozpoczynającej się charakterystyczną sekwencją znaków. Linia nagłówkowa powinna wymieniać nazwy pól logu, oddzielone separatorami. Realistycznym rozwiązaniem w przypadku zmiennego formatu logu byłoby po prostu poprzedzanie nagłówkiem każdej linii logu. Poczynając od linii nagłówkowej, kolejne linie logu będą analizowane zgodnie z opisem w nagłówku aż do napotkania innego nagłówka.

Rozwiązanie dopuszcza agenty heterogeniczne, w tym agenty bez dostępnego kodu źródłowego, o ile tylko pozwalają zapisywać log do pliku i stosują w nim linie złożone z pól rozdzielonych separatorami.

Logi są odczytywane linia po linii (rys. 1), a następnie analizowane zgodnie z separatorami z aktualną definicją z ostatnio napotkanego nagłówka. Dopuszczalne są linie komentarza rozpoczynające się od predefiniowanej sekwencji znaków, które nie są analizowane.

Wartości pól odczytanych z logu są przechowywane w tablicy relacyjnej bazy danych o nazwie log_table (rys. 2). Tablica ta przechowuje poszczególne pola logów jako poszczególne atrybuty. Struktura tej tablicy nie jest znana z góry: liczba i nazwy atrybutów odnoszące się do pól logów są początkowo nieznane. Podczas konsolidacji, w miarę napotykania nowych pól, ta tablica jest rozszerzana przez dodawanie stosownej liczby atrybutów. Nazwy atrybutów i ich typy są przechowywane w tabeli header_table. Typ atrybutu może przyjmować jedną z trzech wartości: liczby całkowitej, liczby rzeczywistej lub łańcucha. Reprezentacje te to zbiory opisane relacją 1.

Choć początkowo nie było to planowane, uzyskana wydajność pozwala rozważać konsolidację logów w czasie rzeczywistym, gdyż osiągnięto wartości około 35 000 pól na sekundę przy stałym formacie logu oraz ponad 20 000 pól na sekundę przy zmiennym formacie. Niemniej przyszłe wersje oprogramowania będą prawdopodobnie wykorzystywały bazy inne niż SQL [17] ze względu na ich większą przydatność podczas modyfikowania tabel i atrybutów.

Address

Artur OPALIŃSKI: Politechnika Gdańska, Wydział Elektrotechniki i Automatyki,
ul. Narutowicza 11/12, 80-233 Gdańsk, Polska, Artur.Opalinski@pg.gda.pl.