Deduction-Based Modelling and Verification of Agent-Based Systems for Data Integration

Radosław Klimek, Łukasz Faber, and Marek Kisiel-Dorohinicki

Abstract. The work concerns the application of multi-agent systems to heterogeneous data integration, and shows how agent approach can be subjected to formal verification using a deductive approach. Since logical specifications are difficult to specify manually, a method for an automatic extraction of logical specifications, considered as a set of temporal logic formulae, is proposed. A simple example is provided.

Keywords: multi-agent systems, formal verification, deductive reasoning, activity diagrams, workflows patterns, temporal logic.

1 Introduction

The growing importance of gathering and analyzing vast amounts of information leads nowadays towards the construction of systems that perform various case-oriented tasks with respect to data coming from numerous, often heterogeneous sources. For example, in [7] authors present a middle-ware to integrate information from heterogeneous enterprise systems through ontologies stored in the XML standard. Apart from providing the semantic data integrity, [1] proposes a way to integrate data sources also on the level of operations. A similar idea of integrating applications, which encapsulate databases, rather than pure databases themselves is presented in [5].

This contribution is based on an agent-based framework dedicated to acquiring and processing distributed, heterogeneous data collected from the various Internet sources [8]. Data processing in such a system is structuralized by means of dynamic workflows emerging from agents' interactions. The goal of the paper is to show how

Radosław Klimek · Łukasz Faber · Marek Kisiel-Dorohinicki
AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: `{rklimek,faber,doroh}@agh.edu.pl`

a formal analysis of these interactions allows to make sure that the system works properly.

In general, formal methods enable the precise formulation of important artifacts and the elimination of ambiguity. Unfortunately logical specifications are difficult to specify manually, and it can be regarded as a significant obstacle to the practical use of deduction-based verification tools. That is why a method providing the automation of generation of logical specifications is proposed. Another contribution is an approach which introduces workflow patterns as logical primitives. Temporal logic is used as it is a well-established formalism, which allows to describe properties of reactive systems. The inference process can be based on both the semantic tableaux method, as well as the resolution-based approach [3].

The paper begins with a short description of the system for data integration. In the next section essential logical background is provided together with the method of specification generation based on workflow describing agents' interactions. Last but not least, the scenario of an example application shows how the approach works in practice.

2 Agent-Based Data Integration Infrastructure

The goal of the discussed system is to provide the data- and task-oriented workflow for collecting and integrating data from a wide range of diverse services. The user is separated from actual data providers by an abstract type system and agents that operate on it.

Tasks created by the user are put into the agent system that performs two types of operations: management of the workflow (by inspecting both data and tasks, and delegating them to other agents) and execution of demanded actions (including their selection, configuration and fault recovery). The system allows to divide processing into *issues*. An issue is intended to be a separate part of data processing, usually focused on some piece of data. An issue is usually created by the user.

The current implementation defines three possible roles for agents.

- System agents: providing basic system functionality, like realisation of new issues, error handling, monitoring (represented by `ControlAgent` in the diagram).
- Issue agents: responsible for keeping track of a single issue and delegating tasks to action agents on the basis of their capabilities. Such an agent retrieves a task and related data from the pool, and explicitly requests a chosen action agent to perform an action specified in the task.
- Action agents: implementing the actual execution of actions. Upon receiving the task from an issue agent, they locate a strategy that can be used to fulfil it and then execute it on data bound to the task. This way they perform any operation over data they receive: merge, simplify, verify etc.

Both issue and action agents provide some description available to other agents, so as to be easily distinguishable in the system. The former are identified by a runtime-generated issue descriptor that represents a topic they are taking care of.

The latter are described in terms of tasks they can perform (called "capabilities") and data types they can operate on.

3 Deduction System

Temporal logic is a formalism for the specification and verification of systems [10]. *Temporal logic* TL introduces symbolism [4] for representing and reasoning about the truth and falsity of formulas throughout the flow of time taking into consideration changes to their values as well as providing information about time structures. Two basic operators are $\Diamond$ for "sometime (or eventually) in the future" and $\Box$ for "always in the future" which are dual operators. The attention is focused on *linear-time temporal logic* LTL, i.e. the time structure constitutes a linear and unbounded sequence, and on the *propositional linear time logic* PLTL. Temporal logics and their syntax and semantics are discussed in many works, e.g. [4, 10]. However, considerations in this work are limited to the *smallest temporal logic*, e.g. [2], which is an extension of the classical propositional calculus to the axiom $\Box(P \Rightarrow Q) \Rightarrow (\Box P \Rightarrow \Box Q)$ and the inference rule $|\!- P \Longrightarrow |\!- \Box P$. The following formulas may be considered as examples of this logic: $action \Rightarrow \Diamond reaction$, $\Box(send \Rightarrow \Diamond ack)$, $\Diamond live$, $\Box\neg(event)$, etc.

Let us introduce some basic notions and definitions. An *elementary set* of formulas over atomic formulas $a_{i,i=1,\ldots,n}$ is denoted $pat(a_i)$, or simply $pat()$, as a set of temporal logic formulas $\{f_1, \ldots, f_m\}$ such that all formulas are syntactically correct. The examples of elementary sets are $Pat1(a,b) = \{a \Rightarrow \Diamond b, \Box\neg(\neg a \wedge b)\}$ and $Pat2(a,b,c) = \{a \Rightarrow \neg\Diamond b \wedge \Diamond c, \Box\neg(b \vee c)\}$. The *logical expression* W_L is a structure, similar to the well-known regular expression, which allows to represent complex and nested structures of elementary sets. The example of logical expression is $Seq(a, Seq(Flow(b,c,d), Switch(e,f,g)))$ which shows the sequence that leads to the sequence of a parallel split (flow) and then conditional execution (switch) of some activities.

Workflow patterns are significant for the approach introduced in this work as they enable the automation of the logical specifications generation process. They constitute a kind of primitives which enable the mapping of design patterns to logical specifications. The proposed method of the automatic extraction of logical specifications is based on the assumption that the entire activity diagrams are built using only predefined workflow patterns. In fact, this assumption cannot be recognized as a restriction since it enables receiving correct and well-composed systems. The *Activity diagram* enables modelling workflow activities. It constitutes a graphical representation of workflow showing flow of control from one activity to another. It supports choice, concurrency and iteration. The important goal of activity diagrams is to show how an activity depends on others [9].

Thus, logical properties for all design patterns are expressed in temporal logic formulas and stored in the predefined *logical properties set* P. The predefined and fixed set of patterns consists of the following basic elements $\Sigma = \{Seq, SeqSeq, Flow, Switch, LoopWhile\}$ the meaning of which seems intuitive, i.e. sequence, sequence of a sequence, concurrency, choice and iteration. The logical properties set is equal to $P = \{Sequence(a1,a2) : in = \{a1\}/out = \{a2\}/a1 \Rightarrow \Diamond a2/\Box\neg(a1 \wedge a2)/SeqSeq(a1,a2,a3) : in = \{a1\}/out = \{a3\}/a1 \Rightarrow \Diamond a2/a2 \Rightarrow \Diamond a3/\Box\neg((a1 \wedge a2) \vee (a2 \wedge a3) \vee (a1 \wedge a3))/Flow(a1,a2,a3) : in = \{a1\}/out = \{a2,a3\}/a1 \Rightarrow \Diamond a2 \wedge \Diamond a3/\Box\neg(a1 \wedge (a2 \vee a3))/Switch(a1,a2,a3) : in = \{a1\}/out = \{a2,a3\}/a1 \wedge c(a1) \Rightarrow \Diamond a2/a1 \wedge \neg c(a2) \Rightarrow \Diamond a3/\Box\neg((a1 \wedge a2) \vee (a1 \wedge a3) \vee (a2 \wedge a3))/LoopWhile(a1,a2) : in = \{a1\}/out = \{a1,a2\}/a1 \wedge c(a1) \Rightarrow \Diamond a2/a1\Diamond\neg c(a1) \Rightarrow \neg\Diamond a2/\Box\neg(a1 \wedge a2)\}$. Formulas a_1, a_2 and a_3 are atomic formulas and constitute formal arguments for a pattern. A slash sign separates formulas. $c(a)$ means that the logical condition associated with the activity a has been evaluated and is satisfied. Variables *in* and *out* provides information about activities for a pattern which are the first and the last to be executed, respectively. In other words, they allow to represent the pattern as a whole.

A logical specification is understood as a set of temporal logic formulas. The sketch of the generation algorithm is presented below. The generation process has two inputs. The first one is a logical expression which represents a workflow model. The second one is a predefined set P. The output of the generation algorithm is a logical specification. The sketch of the generation algorithm is given below.

1. At the beginning, the logical specification is empty, i.e. $L := \emptyset$;
2. Patterns are processed from the most nested pattern located more towards the outside and from left to right;
3. If the currently analyzed pattern consists only of atomic formulas, the logical specification is extended by formulas linked to the type of pattern analyzed, i.e. $L := L \cup pat()$;
4. If any argument is a pattern itself, then the logical disjunction of all elements that belong to *in* and *out* sets, is substituted in the place of the pattern;

The example of the algorithm is provided in the Section 5. The architecture of the deduction-based system using the semantic tableaux method is presented in work [6] where web service models expressed in the BPEL language are considered, but this is a completely different area.

4 Sample Application and Scenario

One of the considered use cases of the agent-based framework is collecting of the data about people with the scientific background. We use data both from services providing personal information (e.g. LinkedIn) and from those strictly professional (e.g. DBLP). Although this kind of a use case may look simple, there are enough interesting tasks and problems that can be used to observe the real behaviour of the system.

The base scenario (from the user's point of view) consists of two steps:

1. *Gathering personal data for a specified person from all available sources.*
   The user feeds the system with a query containing a full name of some person: e.g. "Jan Kowalski". As results of such an action is a list of possible matches,

there is a need to choose one (the best) match. It can be done manually or (in future) delegated to an agent that can rate each result and select the best one.

2. *Getting and merging publications lists from selected sources for the chosen person.*
   The user creates a task to obtain publications from available sources (e.g. DBLP). In this case, when using multiple sources, lists must be merged to create a single and complete publications registry.

This scenario is implemented as follows:

- Types like Person and Publication are introduced.
- Action agents performing operations related to types are implemented: Personal Data Agent and Publications Agent.
- The *merge* action can be implemented in two ways: either the Publications Agent can offer a capability to do the merge specifically for this type or there may exist another agent that can perform a general operation that uses a concrete strategy.
- Strategies for each external service were created: Personal Data Search for e.g. LinkedIn, DBLP or SKOS (AGH internal employee database) and Publications Search for DBLP and BPP.

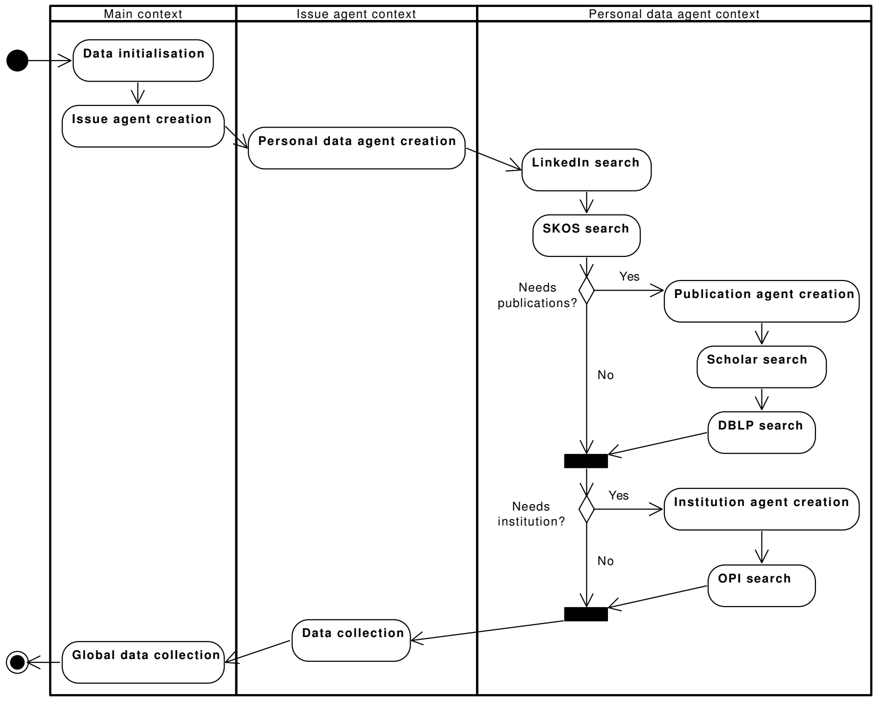

Fig. 1 Activity diagram of the search scenario presented in 4

Figure 1 shows an actual execution of the first step. The user prepares a task specification that consists of a task identifier and initial data to operate on (e.g. a name of the person). The task is placed into the system. Then, all issue agents are notified about it and the one responsible for this task obtains it from the pool. The issue agent locates an action agent that can handle the specified task and delegates its execution to this agent. `PersonalDataAgent` inspects both the task specification and provided data and calls relevant strategies. After that, it sends results (a list of `Person` instances) to the requesting issue agent. It finishes the realisation of the task by putting results to the pool.

5 Formal Analysis of the Scenario

Let us consider the activity diagram shown in Fig. 1. After the substitution of propositions as letters of the Latin alphabet: a – DataInitialisation, b – IssueAgentCreation, c – PersonalDataAgentCreation, d – LinkedInSearch, e – SKOSSearch, f – NeedsPublications, g – PublicationAgentCreation, h – ScholarSearch, i – DBLPSearch, j – NeedsInsitution, k – InstitutionAgentCreation, l – OPISearch, m – DataCollection, and n – GlobalDataCollection, then the expression W_L is

$$Seq(SeqSeq(a,b,c),SeqSeq(Seq(d,e),Seq(Switch(f,\\ SeqSeq(g,h,i),N1),Switch(j,Seq(k,l),N2))),Seq(m,n)) \tag{1}$$

Replacing propositions (atomic activities) by Latin letters is a technical matter and is suitable only for the work because of its limited size. In the real world, the original names of activities are used. Two activities $Null1$ and $Null2$, or $N1$ and $N2$, respectively, are introduced since the diagram in Fig. 1 contains two switches without the else-tasks. The logical expression for the activity diagram is produced in an automatic way.

A logical specification L for the logical expression W_L is built using the algorithm presented in Section 3. The logical specification, which is automatically generated, is

$$\begin{aligned} L = \{&g \Rightarrow \Diamond h, h \Rightarrow \Diamond i, \Box\neg((g\wedge h)\vee(h\wedge i)\vee(g\wedge i)), k \Rightarrow \Diamond l,\\ &\Box\neg(k\wedge l), d \Rightarrow \Diamond e, \Box\neg(d\wedge e), f\wedge c(f) \Rightarrow \Diamond(g\vee i), f\wedge\neg c(g) \Rightarrow \Diamond N1,\\ &\Box\neg((f\wedge(g\vee i))\vee(f\wedge N1)\vee((g\vee i)\wedge N1)), j\wedge c(j) \Rightarrow \Diamond(k\vee l),\\ &j\wedge\neg c(j) \Rightarrow \Diamond N2, \Box\neg((j\wedge(k\vee l))\vee(j\wedge N2)\vee((k\vee l)\wedge N2)),\\ &d \Rightarrow \Diamond e, \Box\neg(d\wedge e), (f\vee i) \Rightarrow \Diamond(N1\vee l\vee N2),\\ &\Box\neg((f\vee i)\wedge(N1\vee l\vee N2)), a \Rightarrow \Diamond b, b \Rightarrow \Diamond c,\\ &\Box\neg((a\wedge b)\vee(a\wedge c)\vee(b\wedge c)), (d\vee e) \Rightarrow \Diamond(f\vee i\vee N1),\\ &(f\vee i\vee N1) \Rightarrow \Diamond(j\vee l\vee N2), \Box\neg(((d\vee e)\wedge(f\vee i\vee N1))\\ &\vee((d\vee e)\wedge(j\vee l\vee N2))\vee((f\vee i\vee N1)\wedge(j\vee l\vee N2))), m \Rightarrow \Diamond n,\\ &\Box\neg(m\wedge n), a \Rightarrow \Diamond n, \Box\neg(a\wedge n)\} \end{aligned} \tag{2}$$

Formal *verification* is the act of proving correctness of a system. Liveness and safety are standard taxonomy of system properties. *Liveness* means that the computational process achieves its goals, i.e. something good eventually happens. *Safety* means that the computational process avoids undesirable situations, i.e. something bad never happens. The liveness property for the model can be

$$c \Rightarrow \Diamond m \tag{3}$$

which means **if personal data agent creation is satisfied then sometime data collection is reached**, formally $PersonalDataAgentCreation \Rightarrow \Diamond DataCollection$. When considering the property expressed by formula (3) then the whole formula to be analyzed is

$$(g \Rightarrow \Diamond h) \wedge (h \Rightarrow \Diamond i) \wedge ... \wedge (a \Rightarrow \Diamond n) \wedge (\Box\neg(a \wedge n)) \Rightarrow (c \Rightarrow \Diamond m) \tag{4}$$

Although the logical specification was generated for only one activity diagram, c.f. formula (2), the method is easy to scale-up, i.e. extending and summing up logical specifications for other activity diagrams and their scenarios. Then, it will be possible to examine logical relationships (liveness, safety) for different activities coming from different activity diagrams.

6 Conclusions

The aim of the paper was to show how a multi-agent system designed for data integration can be subjected to formal verification using a deductive approach. The proposed method based on formal analysis of agents' interactions was illustrated by a simple example. Further research will focus on particular properties of agent interctions in the discussed system. Different application areas will also be considered.

Acknowledgements. The research leading to these results has received funding from the research project No. O ROB 0008 01 "Advanced IT techniques supporting data processing in criminal analysis", funded by the Polish National Centre for Research and Development.

References

1. Agarwal, S., Haase, P.: Process-based integration of heterogeneous information sources. In: Dadam, P., Reichert, M. (eds.) INFORMATIK 2004 - Informatik verbindet (Band 2): Proceedings der 34. Jahrestagung der Gesellschaft für Informatik (GI). Lecture Notes in Informatics, pp. 164–169 (2004)
2. Chellas, B.F.: Modal Logic: An Introduction. Cambridge University Press (1980)
3. Clarke, E.M., Wing, J.M.: Formal methods: State of the art and future directions. ACM Computing Surveys 28(4), 626–643 (1996)
4. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. Elsevier, MIT Press (1990)

5. Hergula, K., Härder, T.: A middleware approach for combining heterogeneous data sources - integration of generic query and predefined function access. In: Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE 2000), vol. 1, pp. 26–33. IEEE Computer Society (2000)
6. Klimek, R.: A deduction-based system for formal verification of agent-ready web services. In: Barbucha, D., Thanh Le, M., Howlett, R.J., Jain, L.C. (eds.) Advanced Methods and Technologies for Agent and Multi-Agent Systems. Frontiers in Artificial Intelligence and Applications, vol. 252, pp. 203–212. IOS Press (2013)
7. Li, S., Zhang, D.H., Zhou, J.T., Ma, G.H., Yang, H.: An xml-based middleware for information integration of enterprise heterogeneous systems. Materials Science Forum 532-533, 516–519 (2006)
8. Nawarecki, E., Dobrowolski, G., Byrski, A., Kisiel-Dorohinicki, M.: Agent-based integration of data acquired from heterogeneous sources. In: International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2011), pp. 473–477. IEEE Computer Society (2011)
9. Pender, T.: UML Bible. John Wiley & Sons (2003)
10. Wolter, F., Wooldridge, M.: Temporal and dynamic logic. Journal of Indian Council of Philosophical Research 27(1), 249–276 (2011)