A Multi-Agent-Based Approach for Autonomic Data Exchange Processes

Hicham Assoudi
LATECE, Département d’Informatique
Université du Québec à Montréal
Montréal, Canada
assoudi.hicham@courrier.uqam.ca

Hakim Lounis
LATECE, Département d’Informatique
Université du Québec à Montréal
Montréal, Canada
lounis.hakim@uqam.ca

Abstract—In this paper, we present a prototype for our solution called Data Exchange Autonomic Manager (DEAM) [1] which has as main goal to turn Data Exchange processes into self-managed systems. We believe that providing data exchange processes with self-healing autonomic capability is a promising approach toward reliable self-managed and resilient data exchange processes. We describe the high level architecture of DEAM prototype which leverages well established techniques and technologies for Autonomic Computing (Multi-Agent Systems) and Schema Matching (Automatic Schema Matching and Mapping).

Keywords—data exchange; autonomic computing; self-managed systems; dependability; fault tolerance; sufficient correctness; schema matching; schema mapping; mapping adaptation; multi-agent systems; agent based modelling and simulation

I. INTRODUCTION

In a previous paper, we proposed a self-healing based approach, called Data Exchange Autonomic Manager (DEAM) [1], to answer the following question: How data exchange processes can be “autonomically” resilient to schema evolution changes? The aim of the proposed solution, DEAM, is to tackle the problem of reducing the human intervention needed to maintain the data exchange processes after a schema evolution [2] (changes impacting source or target system schemas participating in a data exchange scenario). DEAM prototype leverages well established techniques and technologies for Autonomic Computing (Multi-Agent Systems) and Schema Matching (Automatic Schema Matching and Mapping). As far as we know, there is no previous literature describing a solution for self-managed or autonomic data exchange processes.

II. RELATED WORK

Data exchange problem is defined by Arenas et al. [3] as the problem of finding an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source and the target. Data exchange is strongly related to two distinct but complementary concepts: Schema matching and mapping. Those two concepts are often confused and discussed under the single name "Schema Matching" [4].

Schema matching (including its ontology matching variant) has been a very active research area, especially in the last decade, and numerous techniques and prototypes for automatic matching have been developed [5], [6]. Unfortunately, schema matching remains largely a manual and time consuming process [7]. Schema matching has been used as a first step to solve data exchange, schema evolution, or data integration problems [8]. It has also been used for resolving Schema Evolution problems [2] by Maintaining the consistency of mappings under schema changes by finding rewritings that try to preserve as much as possible the semantics of the mappings (Mapping Adaptation [9], [10]).

One of the approaches or paradigms to automate the management of IT solutions, inspired by the nervous system, was proposed by IBM under the name of "Autonomic Computing" [11], [12]. The systems use autonomic strategies and algorithms to handle complexity and uncertainties with minimum human intervention. An autonomic system is a collection of autonomic elements, which implement intelligent control loops to monitor, analyze, plan and execute using knowledge of the environment. One of the main four properties defining an autonomic system is Self-healing. Ghosh et al. [13] provide a definition of self-healing system: “A self-healing system should recover from the abnormal (or “unhealthy”) state and return to the normative (“healthy”) state, and function as it was prior to disruption.”

Agent-Based Modelling and Simulation (ABMS) is a very natural and flexible way to model todays complex, distributed, interconnected and interacting industrial systems: the self-X principles can be mapped on the explicit notion of autonomous agents, modelling a system as a group of interacting agents maps directly onto the distributed and communicative nature of todays systems[14], [15].

III. DEAM PROTOTYPE

This section presents the overall architecture of DEAM and detailed descriptions of agents responsibilities. As we already mentioned in the introduction, DEAM leverages well established techniques and technologies form different research areas:
• Autonomic Computing and Multi-Agent Systems for automatic fault detection, diagnostic and repair (Self-Healing capabilities based on autonomic computing principles);

• Schema Matching for automatic Schema Matching and Mapping (generation of new mapping to recover from mapping degradation after a schema evolution).

In order to implement the autonomic computing principles (i.e. MAPE-K loop) we are leveraging the open source multi-agents platform Jadex which is an extension of the popular and mature java multi-agents platform Jade. Jadex allows the creation of BDI agents supporting the full BDI reasoning cycle with goal deliberation and means-end reasoning [16].

For schema matching, DEAM is leveraging COMA (COMA 3.0 Community Edition). COMA is (previously named COMA and COMA++) is a generic prototypes for schema and ontology matching and it was among the first systems to successfully support the multimatcher architecture and match workflows [17].

A. High-level Architecture

The figure below shows the DEAM architecture, including agents, human actors, and the relationships between agents.

![Figure 1. DEAM High level architecture](image)

The main logical components of DEAM are:

• Data Exchange Scenario Agent: This component represent the data exchange scenario implemented as an active Component in Jadex. Its responsibility is to receive an inbound XML message (from the source system), transform it to the target system format and deliver it to the target system.

• Self-Healing Agent: Jadex BDI agent implementing the MAPE-K loop and responsible for the whole self-healing process. This agent will have to perform the following steps as part of its self-healing process:
  - Detects a failure at the data exchange process and translate it to a standard event format (e.g. Detect a change at the source XSD file after schema evolution).
  - Analyze the event, diagnose a mapping degradation issue (correlation rules) and isolate the broken mappings. During this step the original and the evolved schemas will be compared to capture the changes and to classify them into a set of primitive actions [10] (e.g. adding/deleting elements, merging/splitting elements, etc.).
  - Plan a mapping adaptation by selecting, based on Knowledge Base, the best mapping adaptation approach.
  - Apply the mapping adaptation plan. During this step DEAM will have to modify the data exchange artifacts such as schema definition file and mapping rules file (i.e. XSD, XSLT).

• Benchmarking Agent: This BDI agent provides verification capabilities for the evaluation of the correctness of the healing plans generated by DEAM. It is responsible for automatically generating and injecting perturbations into existing schemas (i.e. DisturbanceInjector) in order to trigger the self-healing agent and evaluate the generated healing plans (i.e. DisturbanceAnalyser: evaluate the correctness of the proposed recovered mappings vs actual correct mappings). The idea behind this agent was inspired from the work done for autonomic benchmarking [18].

• User Agent: This BDI agent is responsible for all the GUI user interactions.

We developed a prototype of DEAM to conduct some preliminary experiments in order to evaluate the idea viability. In the next section, we discuss the experiment example and the obtained results.

B. Prototype Evaluation

This initial version of the prototype implements only the component Self-Healing Agent. In the next releases, we are going to implement the remaining agents: Benchmarking Agent and User Agent. Thus, the experiments conducted so far focuses only on the Self-Healing Agent component (which is the central piece of DEAM's architecture).

To describe the experiment we conducted on DEAM prototype, let's consider the following data exchange scenario. We need to transfer a shipping order between two systems.
The figure above describes the initial mapping between the source and target before the schema evolution for this experiment. After manually introducing a schema evolution (manual modification of the source XSD file to replace the full_name XSD element with two new XSD elements: first_name and last_name), DEAM's monitoring task detected the change and triggered the self-healing process. Below, a summary of self-healing process steps.

- **Step 1**: Monitor a change (Monitor Task)
  - Detect the change (XSD changed after schema evolution)

- **Step 2**: Analyze the change (Analyze Task)
  - Detect the broken mapping (identify, based on the deleted elements the broken mappings)
  - Choose a diagnosis: *Diagnose_Mapping_Degradation*

- **Step 3**: Generate a change plan (Planning Task)
  - Identify the change plan based on Diagnosis *Plan_Mapping_Adaptation*
  - Fetch the plan Plan_Mapping_Adaptation steps

- **Step 4**: Execute the change plan (Execute task)
  - Generate transformation based on the new mapping file

This preliminary version of DEAM's prototype demonstrates, in practical way, the usefulness of the idea of self-healing for data exchange process. As a matter of fact, and as we can clearly see, introducing a change in the source schema resulted in faulty state of the data exchange process which was recovered completely and autonomously (without user intervention) by the autonomic manager (Self-Healing Agent). This experiment results can be considered as encouraging, however we are still working on some interesting challenges such as:

- Developing a new approach/algorithm for Schema Matching in order to replace the Schema Matching tool (COMA3.0) used currently in DEAM prototype.

- Dealing with Complex Mapping problem (Complex mappings map a set of attributes in the source to a set of attributes in the target [19]).

In the next section, we briefly introduce the implementation of our novel Agent-based Modelling and Simulation approach for the Schema Matching problem, called “Schema Matching Agent-based Simulation” (SMAS). We are planning to use our new ABMS approach for Schema Matching to replace the Schema Matching tool (COMA3.0) in the next release of DEAM.

IV. ABMS APPROACH FOR SCHEMA MATCHING

Our solution (SMAS) aims at generating high quality schema matchings with minimum uncertainty. As far as we know, there is no previous literature describing a solution approaching the Schema Matching and Mapping problem under the angle of Agent-Based Modelling and Simulation.

In a nutshell, our idea is to model Schema Matching and Mapping process as interactions between a set of agents (schema attribute agents) within a self-organized environment. Each schema attribute agent belongs to a group (source or target schema attribute) and represents a single schema attribute for which we need to perform the schema matching.

The simulation starts with two groups of unmatched Attribute Agents (source and target groups), which have as main goal the creation of a relation, based on similarity measures, with one of the agents belonging to the other group (for instance, if the agent belongs to the source group it should find the best match in the target group and vice versa).

During each tick of the simulation run, each agent executes behaviors based on randomness (stochastic):

1. Similarity Calculation based on a similarity measures selected randomly from a similarity measures list.

2. Similarity Scores aggregation based on aggregation functions selected randomly from an aggregation function list (MAX, AVERAGE, WEIGHTED).

3. Similarity score validation based on generated random threshold value (within interval)

The simulation ends when each agent has reached a consensus, about its candidate matching, with another agent (both agents are referring to each-other as candidate matching).

SMAS was implemented in Java using the open source ABMS framework “Repast Simphony (2.1)”[20] and the open source framework for Text Similarity “DKPro Similarity (2.1.0)”[21].

We performed preliminary tests of our agent-based model for schema matching for which the results obtained were satisfactory and confirmed our initial intuition about the fact that ABMS can be an efficient and a well suited paradigm for resolving the schema matching problem.

As opposed to deterministic solutions for schema matching, such as COMA, the nondeterministic and stochastic nature of our agent-based simulation increases the confidence in the quality of the matching results. Despite the fact, that the agent's behaviors are based on randomness (e.g. during the similarity calculation), our model can produce, most of the time, the right matchings at the end of each simulation run.

[1] Link for uploaded schema matching simulation animations as well as output files: http://www.researchgate.net/publication/262181441_Schema_Matching_Agent-based_Simulation_(SMAS)_Test_animation
In the next releases of SMAS, we are planning to add statistical analysis in order to produce the final matching results based on multiple simulation runs data (batch mode).

CONCLUSION AND FUTURE WORK

In this work, we described our approach to make data exchange processes resilient to faults resulting from evolving schemas. This preliminary version of DEAM’s prototype is the first step to demonstrate that the idea of self-healing for data exchange process is a promising approach. Also, we briefly introduced the implementation of our novel Agent-based Modelling and Simulation approach for the Schema Matching problem called “Schema Matching Agent-based Simulation” (SMAS). In the next release of DEAM, we are planning to use our new ABMS approach for Schema Matching as replacement for the Schema Matching tool (COMA3.0).

REFERENCES


