Scalable Computing:
Practice and Experience

Scientific International Journal
for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 15(2) June 2014

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 15, Number 2, June 2014

TABLE OF CONTENTS

Special issue on Recent topics in HPC and Clouds:

Introduction to the Special Issue **iii**

Bargain Strategies for Agent Automated Negotiation in an e-Business Environment **121**
Şerban Radu and Adina Magda Florea

Software Modernization and Cloudification using the ARTIST Migration Methodology and Framework **131**
Andreas Menychtas, Kleopatra Konstanteli, Juncal Alonso, Leire Orue-Echevarria, Jesus Gorronogoitia, George Kousiouris, Christina Santzaridou, Hugo Bruneliere, Bram Pellens, Peter Stuer, Oliver Strauss, Tatiana Senkova, Theodora Varvarigou

Adaptive Time-based Coordinated Checkpointing for Cloud Computing Workflows **153**
Bakhta Meroufel and Ghalem Belalem

Regular Papers:

Regulated Condition-Event Matrices for Cloud Environments **169**
Richard M. Wallace, Patrick Martin and José Luis Vázquez

Generalized Matrix Multiplication and its Object Oriented Model **187**
Maria Ganzha, Marcin Paprzycki, and Stanislav G. Sedukhin

DOI 10.12694/scpe.v15i2.978
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 2, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON RECENT TOPICS IN HPC AND CLOUDS

Dear SCPE readers,

It is a pleasure to present a new special issue covering subjects in High Performance Computing and Cloud Computing. Three papers were selected for our current special issue, dealing with various aspects, including strategies for an automated negotiation environment, a methodology for migration and cloudification of legacy software, and a lightweight checkpointing strategy which is adequate to the cloud computing and the workflows characteristics.

Negotiation is an important challenge for different environments, including e-business, grid or cloud environments. Even if different requirements exist for the implementation of the negotiation process in these environments, usually agents-based systems are considered for the implementation of an automated negotiation solution. Even if the results offered by Radu and Florea in [1] are set in relation with an e-business environment, and applied for a couple of usage scenarios, they could be easily expanded for the requirements of clouds.

An interesting intra-server checkpointing strategy is considered by Meroufel and Belalem in [3]. Checkpointing is one of the fault tolerance strategies which can be also used to ensure other services, too. Important issues could be identified in relation with checkpointing and the authors targeting two of them, which are highly relevant in a cloud environment: SLA violations and the increase of system overhead. The proposed Adaptive Time based Coordinated Checkpointing (ATCCp) checkpointing mechanism offers a strong consistency together with the minimum control messages and without communication blocking issues between virtual machines.

A model-driven approach for the modernization and adaptation of legacy applications to cloud environments is considered in the last contribution from this special issue, by Menychtas *et al.* [2]. Developed in the context of the ARTIST FP7 project, the migration methodology identify three major phases, and offer an important set of added value characteristics, including the feasibility analysis in the pre-migration phase, the cloud-compliant aspects, both at SaaS and IaaS level, or the enablement of re-usability and automation.

REFERENCES

[1] Şerban Radu and A. M. Florea, *Bargain strategies for agent automated negotiation in an e-business environment*, Scalable Computing: Practice and Experience, 15 (2014).

[2] A. Menychtas, K. Konstanteli, J. Alonso, L. Orue-Echevarria, J. Gorronogoitia, G. Kousiouris, C. Santzaridou, H. Bruneliere, B. Pellens, P. Stuer, O. Strauss, T. Senkova, and T. Varvarigou, *Software modernization and cloudification using the ARTIST migration methodology and framework*, Scalable Computing: Practice and Experience, 15 (2014).

[3] B. Meroufel and G. Belalem, *Adaptive time-based coordinated checkpointing for cloud computing workflows*, Scalable Computing: Practice and Experience, 15 (2014).

DOI 10.12694/scpe.v15i2.979
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 2, pp. 121–129. http://www.scpe.org

BARGAIN STRATEGIES FOR AGENT AUTOMATED NEGOTIATION IN AN E-BUSINESS ENVIRONMENT

SERBAN RADU AND ADINA MAGDA FLOREA*

Abstract. An automated negotiation environment, in which agents employ different bargaining strategies is described. During negotiation, as more information is exchanged in the negotiation rounds, the agents can change the preferences for certain attributes of the negotiation object. The multi-agent system is developed for a real estate agency business model and several use cases scenarios, using intelligent software agents, are implemented.

Key words: automated negotiation, multi-agent system, negotiation strategy

AMS subject classifications. 68T42, 68T05, 68T35

1. Introduction. Negotiation is an important issue in business environments. Automated negotiation is a process, in which software agents communicate between them, in order to reach a mutually acceptable agreement [1]. An intelligent agent should be able to negotiate with other agents, which have different negotiation behaviors [2]. Best outcome may be obtained if the agent is able to adjust its strategy, predict or guess the strategy of the other agent [3], or choose an adequate strategy, according to the negotiating partner. Different approaches have been proposed, including machine learning approaches, which can be used to change the agent strategy during negotiation, in order to obtain better results and increased payoffs [4].

In our previous works regarding the automated negotiation process, we have proposed bargaining strategies that are based on agent profiles, which can describe statically or can develop dynamically the agent preferences for certain attributes of the negotiation object [5, 6]. Using these profiles, agents obtain better results than in the case when fixed negotiation strategies are employed.

This paper extends our previous work and proposes an agent model, in which the agents apply the negotiation strategy best suited to them, according to the negotiation situation. A multi-agent environment for automated negotiation is designed, offering services for a real estate business model. In the multi-agent system, there are buyer and seller agents and also a facilitator, which is used when a new agent enters into the system, for registering its services. The agents are designed according to the BDI (Belief-Desire-Intention) model [7]. Each agent has a set of goals, selected from the set of desires.

Several use cases scenarios evaluate the negotiation performances. The bargaining takes place in several rounds, before an agreement or a rejection is concluded. In each negotiation round, based on the values and preferences specified by the buyer for the multiple attributes of the negotiation object, the seller agent makes the best possible offer. The agents use linear and non-linear negotiation strategies, which help them in time to increase the gain.

The paper is organized as follows: Section 2 presents the negotiation environment, that is our proposed framework for the negotiation system in an open environment. The agent behavior and how each agent acts during the negotiation process is described in Section 3. In Section 4 is applied the developed negotiation environment for the business model of a real estate agency. The related work is presented in Section 5, while Section 6 discusses conclusions and future work.

2. The Negotiation Environment. The negotiation system consists of cognitive agents, which use a set of negotiation primitives, implemented in the Iterated Contract Net protocol, cf. Fig. 2.1 [8].

The environment is open and the agents are able to enter and leave the environment at any time. The multi-agent system models a heuristic negotiation, in which the agent computes the gain, with respect to its private value for the negotiation object. The agent uses a set of rules to choose the negotiation strategy and to follow that strategy. Also, there is a deadline for the number of negotiation rounds. In a buying negotiation, an agent will look for a lower value than its private value, while in a selling negotiation, its main goal is to obtain more than the item's private value.

*Computer Science Department, University Politehnica of Bucharest, Bucharest, Romania (serban.radu, adina.florea@cs.pub.ro).

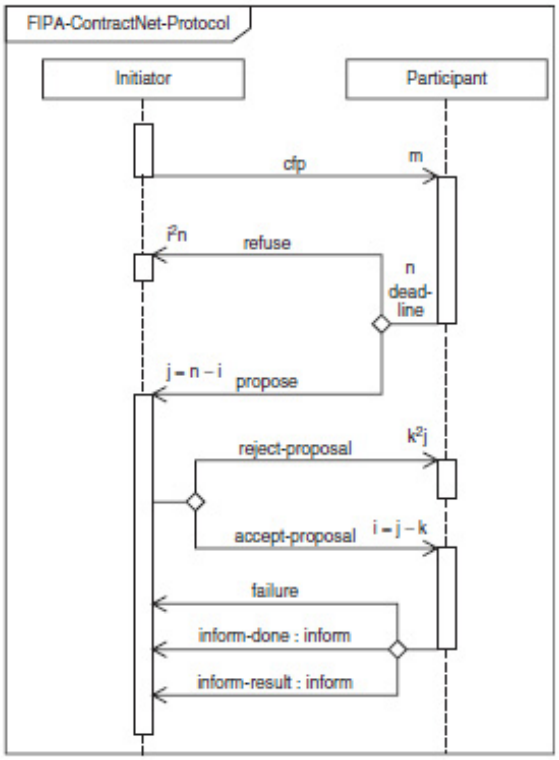

FIG. 2.1. *FIPA-Contract Net protocol. [8]*

The agent behavior is mainly motivated by the gain, but also, depending on a specific context, by the desire to achieve cooperation with other agents. For example, as a general rule, the agent will not accept a price lower than its private value. However, when the agent wants to cooperate, it can accept a lower price. The agent behavior is set up by the negotiation strategy.

The negotiation environment has the following three features [9]:

a) When acquiring multiple goods, a buyer agent only knows the private value available for the set of items, that is the highest price the agent can pay for all the goods, rather than the private value of each separate item;

b) Agents can decommit from tentative agreements at the cost of paying a penalty. Decommitment allows agents to profitably accommodate new negotiations. If these negotiations make some existing contracts less profitable or infeasible for an agent, that agent can decommit from those contracts;

c) Negotiation agents are assumed to have incomplete information about other agents, for example, a buyer agent knows the distribution of the private value of a seller agent and the number of trading competitors. However, an agents negotiation status (the set of proposals it has received) and negotiation strategy are its private information. During negotiation, the agents can quit negotiation at any time, even without notifying their trading partners. When an agent wants to buy multiple goods or services, it concurrently negotiates with sellers to reach agreements for all the items.

In order to evaluate the performance of negotiation agents, a simulation environment consisting of agents negotiating in the business model of a real estate agency, are modeled and implemented in this approach.

In the experiments, agents are using different negotiation strategies, deadlines, and objects to buy or sell. A number of performance measures, such as utility, gain, number of successful negotiations, learning capabilities, are determined.

A negotiation object represents the range of features and issues over which agreements must be reached. In case of single-issue negotiation, the object of negotiation may be either a unique item, for example a good that the agent A_1 wants from A_2 or a service that agent A_1 offers to A_2. In case of multi-issue negotiation, the

negotiation object is replaced by a negotiation package, composed of a set of negotiation objects.

Each agent has two prices, the minimum price and the maximum price, between which it accepts offers. The buyer computes its gain as the difference between the maximum price it is willing to pay and the price of the current offer. The seller computes its gain as the difference between the price of the current offer and the minimum price it is willing to accept.

An agent is focused on a set of useful strategies. These strategies can be predefined or, alternately, can be learned in time, such that to choose the best strategy for negotiating with a certain agent. There are a lot of factors which can influence the result of a negotiation strategy. These factors refer to the strategies of other agents, their constraints and preferences, and other characteristics of the negotiated issues. The strategies are rule-based and are reusable for different negotiation problems, with different agents.

Agents use **linear and non-linear strategies**, with the non-linear strategies divided in: **conceder strategy**, if the agent is willing to concede a lot in the first rounds of negotiation, and **boulware strategy**, if the agent is willing to concede considerably only when the time deadline is close [10].

A seller agent starts by making an offer that represents a high price, and over time, as it makes concessions, the price is decreased. The difference between the non-linear strategies lies in the way in which the price comes down. In the conceder strategy, the agent makes all its concessions early and then doesn't concede much, as the negotiation evolves and the deadline approaches. In the boulware strategy, the agent doesn't initially change the price so much at each time step. But, as the deadline for negotiation approaches, the size of its concessions increases by larger amounts. During the bargaining process, these negotiation strategies are used in our environment, both by buyers and sellers.

Different negotiation scenarios are tested, in an open and adaptive multi-agent system. The framework is implemented using **Java**, **Jade**, **Jess**, and **XML**. The system has the possibility to add or remove agents during run-time, showing that it is an open environment. **Jade** is used as an infrastructure for creating the multi-agent system and **Jess** is a mechanism for providing the inference engine for the agents that are negotiating. The **Jess** engine represents the agent inference engine, which stores the knowledge base of the business domain and also contains the negotiation strategies of the agent, the facts and the concepts of bargaining. Also, **Jess** rules describe the way in which the price is computed during negotiation, with respect to the negotiation strategy of the agent.

During negotiation, an agent wants to maximize its gain, by fulfilling its goals. Also, negotiation criteria refer to the cooperation profile the agent has developed to describe the interaction history with other agents in the system. The cooperation profile is considered as a part of the agent belief about other agents in the system.

3. Agent Behavior. The agents have different reasoning capabilities, used to conduct successful negotiation and to reach their goals. Each agent learns in time which strategy gives the maximum utility [11]. For doing this, each agent has an associated utility values matrix, which helps the agent to choose the strategy which maximizes its utility.

Any agent from the system uses a set of behavior rules, which define how the agent fulfills the goals, and a set of strategy rules, which guides the negotiation process.

The negotiation strategy of an agent may show how much and how quickly the prices are decreased and if the price is lower than the private value of the agent. An agent can use a tradeoff in negotiation. In the case of cooperative agents, an agent can gain more once, and then can sell cheaper. This is a kind of global evaluation on previous deals.

The negotiation strategy is implemented in the form of rules, each agent keeping a history of its interactions. If in a given situation, several rules are eligible, then the negotiation strategy decides which rule, from the conflict set, to be applied. A possible approach to solve the conflicts is to assign priorities between rules. The solution is to apply the rule with the highest priority. The priority of the rules is dynamically modified, according to the negotiation situation.

The agents in the system use different strategies, for instance, at each negotiation step, the price is decreased by 1, or is decreased by 3. Another strategy rule tells what happens when the price increases with 10% above the private value, if the offer is instantly accepted or not.

In our research, the cooperation profile describes the preferences about the agents with which an agent wants to cooperate. This profile is implemented as a dynamic structure. For each agent with which a negotiation is

performed, the knowledge regarding the negotiation result is stored in the cooperation profile. This is updated during the negotiation process, at the end of each negotiation.

The classification of the partner agent represents the current agent belief about the cooperation potential of the partner. The partners are classified into six cooperation classes: highly cooperative, very cooperative, cooperative, slightly cooperative, non-cooperative, and unknown [5]. While more negotiations take place, the cooperation class associated to the partner agent can be changed. According to the negotiation history with a certain agent and depending on the cooperation class of the partner, the agent learns in time which negotiation strategy offers the maximum gain.

The characterization of the cooperation potential of a partner agent is done by classifying the partners into cooperation classes, which divides the cooperation ability of the partner into six classes. The classification is done using the C4.5 learning algorithm, in which a decision tree is a classifier for the cooperation degree, expressed as a recursive partition of the instance space. In the decision tree, the class is represented by the partner classification field. During the negotiation rounds, the C4.5 algorithm can classify the partner agent into another cooperation class, if the values of the attributes used in the algorithm are changed.

In the statistics file, associated to the multi-agent system, the following information is collected:

a) **Negotiation Statistics** - for each agent, the system collects the total gain, the number of negotiations, and the total number of negotiation rounds;

b) **Cooperation Statistics** - for each agent and partner cooperation class, the system collects the number of negotiation rounds, the number of negotiations, the total gain, and the list of agents included in that cooperation class;

c) **Supply-Demand Statistics** - supply is computed as the sum of quantities of all seller products and demand is computed as the sum of quantities of all buyer products. Their ratio is computed by dividing the demand to the supply.

In the implemented system, developed in this research, there is a **base agent class**, extended by buyer and seller agents, which contains common functions used by all agents. The main features of this class are described in the following steps:

1. *Read the negotiation object*;
2. *Check* ***if*** *the current agent has objects with the desired attributes*;
3. ***If*** *there are no objects in the stock* ***then*** **REJECT-PROPOSAL**;
4. *Check the message type*;
5. ***If*** *message*=**CALL-FOR-PROPOSAL** (only sellers receive CFP) ***then*** *get all objects of that type*;
   5.1. ***If*** *there are less objects than required* ***then*** **REJECT-PROPOSAL**;
   5.2. ***If*** *there are offers* ***then*** *send them*;
6. ***If*** *message*=**INFORM** (only buyers receive INFORM) ***then***
   6.1. ***If*** *multiple offers are received* ***then*** *find which one is the best*;
   else *check* ***if*** *the attributes' values match the request*;
7. ***If*** *message*=**PROPOSE** ***then*** *process offer*;
8. ***If*** *message*=**AGREE** ***then***
   8.1. *Add information to statistics file and compute gain*;
   8.2. *Remove object from stock*;
   8.3. *Collect statistics when negotiation ends*;
9. ***If*** *message*=**REJECT-PROPOSAL** ***then*** *end negotiation with reject.*

Each **buyer agent** does the following steps:

1. *Every 5 seconds the buyer agent sends to all sellers* **REQUESTS** *for the objects of interest*;
2. *Get the agents which sell what the agent needs to buy.*

Each **seller agent** performs the following steps, when processing an **ACCEPT** offer:

1. *Remove the object from the stock*;
2. ***If*** *agreement on price is set* ***then*** *sends* **ACCEPT** *message to the buyer agent*;
3. *Add the negotiation results to statistics and compute the gain*;
4. *Remove the current bid from the list of open bids*;
5. *Reject the offers of other agents interested in this object*;

TABLE 4.1
The Buyer Agents Requirements and the Seller Offer in the Three-to-One Automated Negotiation Scenario, with Two House Types.

Agent	House Type	Quantity	Price
B_1,B_2,B_3	2 Rooms	5	40000-60000
B_1,B_2,B_3	3 Rooms	5	80000-100000
S_1	2 Rooms	20	45000-70000
S_1	3 Rooms	20	85000-110000

6. *Add to statistics file the result of the failed negotiations.*

4. Real Estate Agency Automated Negotiation Business Model. An automated negotiation environment between real estate agencies and real estate developers is realized. The houses to be sold have different attributes, expressed in the **XML configuration file**, associated to this business model. The negotiation is done based on price, but there are also other attributes in the configuration file, such as the number of rooms, rooms' dimensions, quality of finishes, location of the house.

The negotiation process starts when the buyer agent wants to buy a house. A request for the house is sent to all sellers. These agents respond to the request with their configuration for the house. After receiving the initial offers, the buyer evaluates and compares different offers. When an offer is not satisfactory, the buyer agent makes a counterproposal to the corresponding seller agent. This has a set of strategies that configure its constraints. The particular strategy used in a certain moment of negotiation depends on the market dynamics and can be changed from the graphical interface associated with each agent. When the strategy is changed, the suitable **Jess** rules are applied by the agent.

The buyer agent from the real estate agency scenario has several criteria, represented by attributes, each having a certain priority for the user. These characteristics are encoded in the **XML** configuration file, associated to the business model. The content of the configuration file is read into the application using the **SAX** parser.

The rules, upon which the negotiation is performed, are defined in **Jess**. There are different rules defined for each type of the communication primitive. The higher priority is associated to the **ACCEPT** rules, the medium priority to the **REJECT** rules, and the lower priority to the **PROPOSE** rules.

The negotiation is ended when one of the agents sends an **ACCEPT** message, or when the time expires, and a **REJECT** message is concluded. There are different rules associated to each communication primitive of the agents, and also for each strategy of the agent.

Some examples of **Jess** rules, upon which the negotiation is performed in the real estate agency business model, are described in what follows:

a) The first **ACCEPT** rule says that an offer for a quantity q for a certain product, having the price between *minPrice* and *maxPrice*, from an unknown or non-cooperative partner, is accepted, if the price is less than *q*(minPrice+maxPrice)/2*;

b) When using the **conceder strategy**, the buyer increases its previous offer with *800*, towards very cooperative or highly cooperative sellers, when the negotiation step is greater than *12*;

c) When using the **boulware strategy**, the seller decreases its previous offer o by *2000*, when the negotiation step is greater than *1* and the time elapsed in the negotiation is greater than *4.5* seconds, represented by the global variable *boulwareTime2*;

4.1. Three-to-One Automated Negotiation Scenario. The following scenario involves a real estate agency. There are three buyers, wanting to buy houses from the real estate agent, each buyer having a different negotiation strategy: B_1-linear, B_2-conceder, and B_3-boulware. The seller S_1 has a linear strategy. The requirements of the agents are represented in Table 4.1.

Figure 4.1 represents a screen capture of the **Jade Sniffer Agent** for this scenario, showing the messages exchange between agents.

The next diagrams (cf. Figs. 4.2 and 4.3) are obtained using the data collected in the statistics file, generated after the negotiation is performed between the agents in the platform. When all the buyers finish their purchases, the negotiation information is recorded in the statistics file.

The buyers gain, obtained after negotiation, is represented in Fig. 4.2.

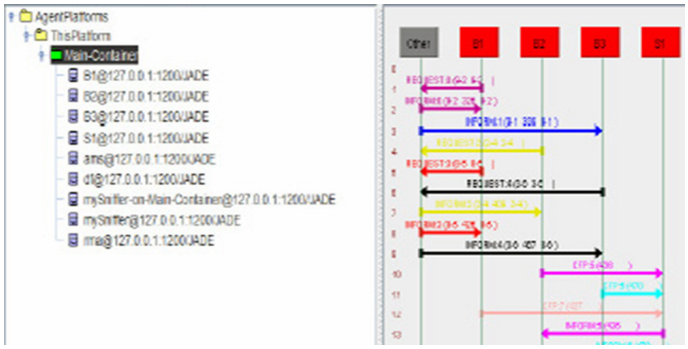

FIG. 4.1. *Screen capture showing the messages exchange between three buyers and one seller.*

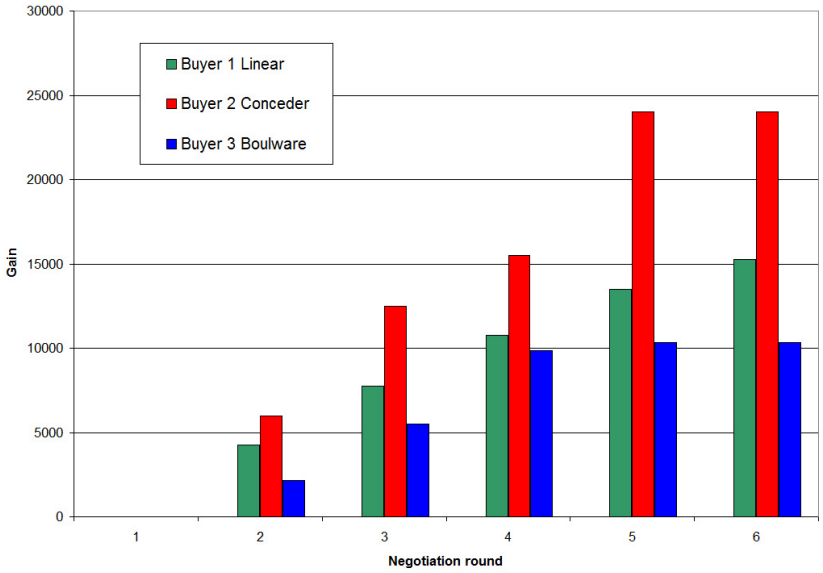

FIG. 4.2. *The three buyers gain, each using a different negotiation strategy.*

With respect to the gain of the buyers obtained after negotiation, the higher gain is obtained for the buyer using the conceder strategy, the medium gain corresponds to the linear strategy and the smaller gain is for the boulware strategy.

4.2. Two-to-Three Automated Negotiation Scenario. The next scenario involves two buyers and three sellers, having the requirements displayed in Table 4.2. The buyers use a linear strategy, but the sellers are using different strategies: S_1 has a linear strategy, S_2 a conceder strategy, and S_3 a boulware strategy.

Figure 4.3 represents the sellers gain versus the negotiation rounds. Each seller uses a different strategy during negotiation.

Regarding the gain of the sellers obtained after negotiation, the higher gain is obtained for the seller using the boulware strategy, the medium gain corresponds to the linear strategy and the smaller gain is obtained for the conceder strategy.

5. Related Work. There are some different approaches in the automated negotiation domain. A negotiation system is presented in [12], which supports the design of different strategies for agent negotiation, and the evaluation of these strategies in a simulated environment. The designer of a strategy can select from a repository, a negotiation domain and a preference profile for the agent. As compared to this system, in our

TABLE 4.2
The Buyer Agents Requirements and the Sellers Offer in the Two-to-Three Automated Negotiation Scenario, with Three House Types.

Agent	House Type	Quantity	Price
B_1	2 Rooms	5	40000-60000
B_1	3 Rooms	5	80000-100000
B_1	4 Rooms	5	100000-120000
B_2	2 Rooms	5	45000-65000
B_2	3 Rooms	5	85000-105000
B_2	4 Rooms	5	105000-125000
S_1	2 Rooms	10	50000-70000
S_2	3 Rooms	10	90000-110000
S_3	4 Rooms	10	110000-130000

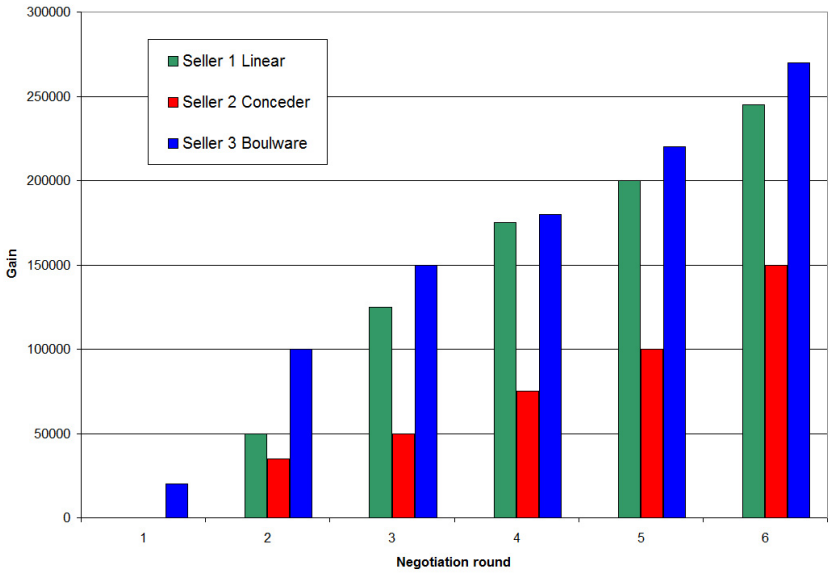

FIG. 4.3. *The three sellers gain, each using a different negotiation strategy.*

approach, there are several negotiation strategies, which can be changed during negotiation, and the negotiation domain is specified by the agent rules.

A negotiation strategy that describes a method to learn a model of opponent preferences in a single negotiation session is presented in [13]. The negotiation strategy should be efficient, transparent, maximizing the chance of an agreement and should avoid exploitation. In the current paper, we have applied different negotiation strategies, in order to improve in time the behavior of the agents.

A model of iterative reasoning process is developed in [3], by widening the notion of a level in a hierarchy, from one single strategy to a distribution over strategies, leading to a more general framework of multi-agent decision making. Our approach combines different strategies, in order to find the best possible outcome.

The impact of the negotiation environment on the performance of several intra-team strategies is studied in [14]. An agent-based negotiation team is a group of agents that joins together, because they share common interests in the negotiation. In our approach, the agents are self-interested and are using a set of strategies, towards achieving their goals, trying to obtain the maximum gain.

An agent, which learns the general pattern of behavior, based on all of the interactions in which it participates is developed in [15]. A generic approach, which may help the agent compete against unknown opponents in different environments is proposed. In our research, the agents' beliefs about the other agents in the system are combined with the possibility to represent and modify dynamically the negotiation strategy.

The decision to accept an offer is presented as a sequential decision problem in [16], by modeling the bids received as a stochastic process. This choice is useful in the context of negotiation with incomplete information, where the future behavior of the opponent is uncertain. In our approach, the agents use the Iterated Contract Net protocol, which has the advantage that it can simulate a real-world scenario, with many buyers and sellers having parallel negotiations.

A multi-issue negotiation that runs under real-time constraints and in which the negotiating agents have no prior knowledge about their opponents' preferences and strategies is described in [17]. A negotiation strategy, which employs sparse pseudo-input gaussian processes to model efficiently the behavior of the negotiating opponents, is used. In our approach, the strategy is represented in the form of rules, with their attached preference coefficients.

An agent, which estimates the opponent's strategies based on the past negotiation sessions, is presented in [18]. The agent tries to compromise to the estimated maximum utility of the opponent by the end of the negotiation. In our system, the agents' behavior can change during negotiation, according to previous interactions with other agents. Changing behavior may refer to either the use of different negotiation strategies or to concessions made for other agents, with which they have successfully negotiated in the past.

Modeling the decision-making process, through the design of a goal directed negotiation model, based on the BDI agent theory, is developed in [19]. A prototype of the model is built and applied to an aircraft purchase negotiation process. Also, in our system, the agents are designed according to the BDI model. Automated negotiation agents are developed, having adaptive negotiation strategies. The model is implemented and the agent behavior is tested on different settings, for a real estate agency business model.

6. Conclusions and Future Work. A multi-agent system for automated negotiation, involving a real estate agency business model, was presented. The gain obtained by agents during negotiation is computed after each negotiation round and is graphically represented.

The negotiation strategy of the agents can be changed in time, as more negotiations are taking place. The agents have a set of useful negotiation strategies, from which they can choose the best one. The three possible negotiation strategies: linear, conceder, and boulware, can be dynamically changed, during run-time, for each negotiating agent, being either buyer or seller, using the graphical interface of each agent. The combination of strategies for buyer and seller agents gives different gains for the agents.

The experiments demonstrate distinct behavior and gain for the agents employing different strategies. The strategy used by agents is dependent on the number of buyers and sellers in the virtual market. Also, the tests performed showed the improvement of the agents performance in time, with respect to their negotiation abilities.

Because the agents preferences are based on their goals, changes of their needs influence the preferences during negotiation. When new information is available, the agents can update their preferences regarding the negotiation results. The communication primitives and the framework can express complex negotiation dialogues, in which agents can change their preferences in time.

Heuristic negotiation strategies used in this article are based on the exchange of proposals. In the case the answer received from the partner agent is a counterproposal, the argumentation-based negotiation extends the negotiation protocol with the possibility to exchange arguments. This information gives explicitly the opinion of the agent making the argument.

Future work will investigate the arguments used by agents for improving the negotiation outcomes. Also, future work will be directed towards the implementation of more complex agent strategies and knowledge sharing ability between agents.

Acknowledgments. This work was supported by the project ERRIC No. 264207, FP7-REGPOT-2010-1.

REFERENCES

[1] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, M. J. Wooldridge, and C. Sierra, *Automated Negotiation: Prospects, Methods and Challenges*, International Journal of Group Decision and Negotiation, 10, 2 (2001), pp. 199–215.

[2] S. S. Fatima, M. J. Wooldridge, and N. R. Jennings, *On Optimal Agendas for Multi-Issue Negotiation*, Proceedings of the 12-th International Workshop on Agent-Mediated Electronic Commerce, 2010, pp. 155-168.

[3] M. Wunder, M. Kaisers, J. R. Yaros, and M. Littman, *Using Iterated Reasoning to Predict Opponent Strategies*, Proceedings of the 10-th International Conference on Autonomous Agents and Multiagent Systems, 2 (2011), pp. 593–600.
[4] S. Manju and M. Punithavalli, *An Analysis of Q-Learning Algorithms with Strategies of Reward Functions*, International Journal on Computer Science and Engineering, 3, 2 (2011), pp. 814–820.
[5] S. Radu, E. Kalisz, and A. M. Florea, *A Model of Automated Negotiation Based on Agents Profiles*, Scalable Computing: Practice and Experience Journal, 14, 1 (2013), pp. 47–55.
[6] S. Radu, E. Kalisz, and A. M. Florea, *Automatic Negotiation with Profiles and Clustering of Agents*, International Journal of Intelligence Science, 3, 2 (2013), pp. 69–76.
[7] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. J. Wooldridge, *The Belief-Desire-Intention Model of Agency*, Intelligent Agents V: Agents Theories, Architectures and Languages, Lecture Notes in Computer Science, 155 (1999), pp. 1–10.
[8] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, John Wiley & Sons Ltd, UK, 2007.
[9] B. An, V. Lesser, and K. M. Sim, *Strategic Agents for Multi-Resource Negotiation*, Autonomous Agents and Multi-Agent Systems, 23, 1 (2011), pp. 114–153.
[10] M. Wooldridge, *An Introduction to MultiAgent Systems*, Second ed., John Wiley & Sons Ltd, UK, 2009.
[11] S. Radu and V. Lungu, *An Adaptive Multi-Agent Model for Automated Negotiation*, Proceedings of the 19-th International Conference on Control Systems and Computer Science, 1 (2013), pp. 167–174.
[12] R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry, *Negotiation with Bounded Rational Agents in Environments with Incomplete Information using an Automated Agent*, Journal of Artificial Intelligence, 172 (2008), pp. 823–851.
[13] K. Hindriks, C. M. Jonker, and D. Tykhonov, *The Benefits of Opponent Models in Negotiation*, Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, 2 (2009), pp. 439–444.
[14] V. Sanchez-Anguix, V. Julian, V. Botti, and A. Garcia-Fornes, *Studying the Impact of Negotiation Environments on Negotiation Teams' Performance*, Information Sciences, 219 (2013), pp. 17–40.
[15] R. Azoulay, R. Katz, and S. Kraus, *Efficient Bidding Strategies for Cliff-Edge Problems*, Journal of Autonomous Agents and Multi-Agent Systems, 28 (2014), pp. 290–336.
[16] T. Baarslag and K. V. Hindriks, *Accepting Optimally in Automated Negotiation with Incomplete Information*, Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems, 2013, pp. 715–722.
[17] S. Chen, H. B. Ammar, K. Tuyls, and G. Weiss *Optimizing Complex Automated Negotiation using Sparse Pseudo-Input Gaussian Processes*, Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems, 2013, pp. 707–714.
[18] K. Fujita *Automated Strategy Adaptation for Multi-Times Bilateral Closed Negotiations*, Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems, 2014, pp. 1509–1510.
[19] M. Cao *Goal Deliberated Agent Architecture for Multi-Strategy Selection in Automated Negotiation*, Proceedings of the 14-th Annual International Conference on Electronic Commerce, 2012, pp. 159–160.

Edited by: Florin Fortiş
Received: Mar 1, 2014
Accepted: Jun 24, 2014

DOI 10.12694/scpe.v15i2.980
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 2, pp. 131–152. http://www.scpe.org

SOFTWARE MODERNIZATION AND CLOUDIFICATION USING THE ARTIST MIGRATION METHODOLOGY AND FRAMEWORK

ANDREAS MENYCHTAS[1], KLEOPATRA KONSTANTELI[2], JUNCAL ALONSO[3], LEIRE ORUE-ECHEVARRIA[4], JESUS GORRONOGOITIA[5], GEORGE KOUSIOURIS[6], CHRISTINA SANTZARIDOU[7], HUGO BRUNELIERE[8], BRAM PELLENS[9], PETER STUER[10], OLIVER STRAUSS[11], TATIANA SENKOVA[12], AND THEODORA VARVARIGOU[13]

Abstract. Cloud computing has leveraged new software development and provisioning approaches by changing the way computing, storage and networking resources are purchased and consumed. The variety of cloud offerings on both technical and business level has considerably advanced the development process and established new business models and value chains for applications and services. However, the modernization and cloudification of legacy software so as to be offered as a service still encounters many challenges. In this work, we present a complete methodology and a methodology instantiation framework for the effective migration of legacy software to modern cloud environments.

Key words: Cloud Computing, Legacy Software, Modelling, Migration, Methodology, SaaS

AMS subject classifications. 68M14, 68U01, 68U35, 90B25

1. Introduction. Nowadays, cloud computing [4] appears as one of the most popular and mature technological and business environments for engineering, hosting and provisioning software applications. A continuously increasing set of cloud-based solutions across the cloud stack layers [11] is available to application owners and developers to tailor their applications and exploit the advanced features of this paradigm for elasticity, high availability and performance. These solutions provide many benefits to new applications but they also introduce constrains to the modernization and migration of legacy applications. We consider *legacy applications* as software not developed for the Cloud and software in traditional architectural paradigms that cannot be scaled, cannot be measured and does not share resources beyond infrastructure (e.g. database, memory). Often the legacy applications follow monolithic architecture design approaches, implemented in technologies which may be deprecated or cannot easily deal with the notion of "as a Service" and are installed on owned infrastructures.

The modernization and adaptation of legacy applications to cloud environments is a great challenge for all involved stakeholders, not only from a technical perspective, but also in business level with the need for adaptation of the business processes and models of the application which will be deployed on the Cloud and offered "as a service". In this paper, we present a novel model-driven [22] approach for the migration of legacy applications in modern cloud environments which covers all aspects and phases of the migration process, as well as an integrated framework that supports all migration process.

Our motivation for this work is the requirements and challenges for the effective migration of legacy software on the Cloud as described in [9]. To this end, the proposed migration methodology considers the following aspects:

- **Unknown internal structure** due to the complexity of the software and the data management processes.
- **Lack of knowledge for target environment** where the application will be deployed and provisioned.

[1]National Technical University of Athens (ameny@mail.ntua.gr)
[2]National Technical University of Athens (kkonst@mail.ntua.gr)
[3]Tecnalia Research & Innovation (juncal.alonso@tecnalia.com)
[4]Tecnalia Research & Innovation (leire.orue-echevarria@tecnalia.com)
[5]ATOS Research & Innovation (jesus.gorronogoitia@atosresearch.eu)
[6]National Technical University of Athens (gkousiou@mail.ntua.gr)
[7]National Technical University of Athens (csantz@mail.ntua.gr)
[8]Inria, Mines Nantes & LINA (hugo.bruneliere@inria.fr)
[9]Spikes N.V. (bram.pellens@spikes.be)
[10]Spikes N.V. (peter.stuer@spikes.be)
[11]Fraunhofer Institute for Industrial Engineering IAO (oliver.strauss@iao.fraunhofer.de)
[12]Fraunhofer Institute for Industrial Engineering IAO (tatiana.senkova@iao.fraunhofer.de)
[13]National Technical University of Athens (dora@telecom.ntua.gr)

- **Multi-tenancy influence** which creates a number of issues that span from security to performance and availability.
- **Variable configuration based on user preferences** which poses additional customization requirements when the application is offered as a service.
- **Various application types** with different characteristics and usage of resources e.g. networking, storage etc.

Unlike current methodologies, in this work we present an end-to-end approach developed in frame of ARTIST EU Project [1] that covers both all migration and modernization phases and also considers both the technical and business aspects of the application. This complete model-driven modernization and migration approach (initially discussed in [12]) is supported by an innovative toolbox as a "one stop shop" for the migration, modernization and cloudification of legacy applications. ARTIST Migration Methodology introduces three phases for the migration of legacy applications to cloud environments starting from a feasibility analysis to the core modernization of the software and its validation and certification during the post-migration. All aspects of the migrated software are examined in the methodology including the migration of the data and the adaptors which may be required for the handover between the legacy data source and the modern cloud data stores.

In addition, the proposed framework includes a rich set of tools, which realize each phase, task and activity of the methodology. The methodology and the framework provide a complete environment where all the stakeholders involved in the migration of an application (analysts, developers, engineers etc.) can collaborate under well defined workflows and processes that are customized and instantiated for the particular migration project. The post-migration aspects are also considered by defining specific processes for the behavioural equivalence (functional and non-functional parameters) and the reuse of software artefacts.

The rest of the paper is structured as follows: Section 2 highlights the related work in this field while Section 3 describes in detail the proposed migration methodology. Section 4 analyses the architecture design of the overall framework that supports the methodology and Section 5 the implementation details of the Methodology Process Tool, which is the core component for the customization and instantiation of the methodology. An evaluation of the proposed solution based on an experimental application is described in Section 6. Finally in Section 7, the conclusions of our work are presented.

2. Related Work. Prior to the definition of the various methodology elements, we examined the related migration approaches and extracted some interesting findings, with the most important one the fact that there is no methodology covering all migration processes/phases required in our approach.

G. Lewis et al. [10] proposed the SMART methodology which is relevant to the pre-migration phase while trying to understand the system operation in order to be able to identify and analyse the gap between the legacy system and the desired one. However, most of this analysis is performed manually and based on the knowledge of the participating team. After acquiring that knowledge, SMART proposes an ad-hoc migration strategy created for each system individually. In addition, since SMART is focused on the migration to SOA, many relevant issues that concern the basics of SaaS architectural design are not treated.

The Butterfly method [26] guides the migration of a mission-critical legacy system to a target system and consists of 5 phases, namely: *justification, legacy system understanding, target system understanding, migration* and *testing*. Justification phase involves the investigation of the risk and benefits associated with the legacy system evolution while legacy system understanding involves reverse engineering of the legacy system in order to identify the components, recreate documentation, understand the static and dynamic behaviour of the legacy system. Target system development involves elicitation of requirements/specifications of the target system and choosing the most appropriate architecture and standards. The migration phase is concerned with the physical transformation of the whole legacy system to the target system.Finally, testing is carried out throughout the evolution process to ensure that the target system delivers the functionalities specified at the start of the evolution.

Warren and Ransom [25] developed the Renaissance method which examines a legacy system from all technical, business and organizational perspectives. The method guides users through assessment of these perspectives by selecting assessment characteristics and assigning values to them. According to this approach system assessment is used to gain an understanding of a legacy system, which is fundamental to any system evolution exercise. System assessment should be an initial activity for evolution projects. The first important

milestone in the Renaissance method is a viable, cost effective system evolution strategy which can be presented to higher management. The activity of assessing the current system is supported by another activity for modelling which is focused on increasing the level of knowledge about the system on a conceptual level.

OMG ADM (Architecture-Driven Modernization) [16] provides a set of generic metamodel specifications that could be relevant within the context of our work. The Software Measurement Metamodel (SMM) proposed by this methodology, could be used in the assessment of the migration process (e.g. for expressing appropriate metrics/measures as well as representing the result of their computation), the Knowledge Discovery Metamodel (KDM) and to a lower extent the Abstract Syntax Tree Metamodel (ASTM) can be exploited for reverse engineering purposes (e.g. for representing the legacy source code as models in a neutral technology-independent manner).

A. Bagnato et al. [2] implemented the XIRUP, which is considered as a general-purpose MDE-based modernization methodology, not specifically designed to address the challenges of migrating legacy applications to cloud environments. In particular, XIRUP is a feature-driven modernization methodology, where the whole legacy system is decomposed into features (as offered by encapsulated components) that are iteratively evaluated (for migration decision support), migrated and assessed (post-migration evaluation). Nonetheless, even if some of these method fragments are applicable to foreseen methodology phases, they need to be aligned and extended to cope with the particularities of the migration to the Cloud.

Model Driven Reverse Engineering (MDRE) itself, as the application of Model Driven Engineering (MDE) principles and techniques to reverse engineering problems, is a rather recent area [21]. However, there have been really few generic and extensible MDRE approaches proposed so far. The MoDisco two-step Model Discovery + Model Understanding approach and corresponding framework in Eclipse [3] is one of them. As covering the reverse engineering phase, it can be directly reused and then extended when necessary within the context of the more global approach.

The REMICS methodology proposed by Mohagheghi and Sæther [15], while following an MDE approach, does not take into consideration non-functional requirements (i.e. performance) inherent to SaaS applications and neither addresses architectural issues such as multi-tenancy or scalability. REMICS relays the monitoring, billing and security issues to the cloud provider where the application is deployed on. The business issues (business model and processes), related to the components mentioned before (billing, monitoring) are also not considered. In addition, REMICS executes the migration on brute force, without considering the feasibility or convenience to migrate.

In the following table we summarize the various aspects of the migration methodologies defining the baseline for the ARTIST Migration Methodology:

Table 2.1
Migration Methodologies Comparison

	Business Aspects	Technical Aspects	Cloud Aspects	Migration Assessment	Validation & Verification
SMART	✘	✔	✘	✔	✔
Butterfly	✔	✔	✘	✔	✔
Renaissance	✔	✔	✘	✔	✘
OMG ADM	✘	✔	✘	✔	✘
XIRUP	✔	✔	✘	✔	✔
MoDisco	✘	✔	✘	✘	✔
REMICS	✘	✔	✔	✘	✔

3. ARIST Migration Methodology.

3.1. Methodology Overview. The methodology was developed to cover all aspects of software migration and modernization, including the business and technical requirements posed from the modern cloud ecosystems as common target environments for the deployment and provisioning of applications. The methodology consists of three (plus one) major phases which are analysed below:

These three main phases are:

- **Pre-migration:** In this phase a study of the technical and economic feasibility will be conducted as a prerequisite to the migration/modernization of the legacy system so as to effectively define the exact steps and effort that will be required.
- **Migration and Modernization:** This phase will perform the migration process itself, by using both reverse engineering (RE) and forward engineering (FE) techniques in order to deploy the legacy system on the Cloud. It should be noted that based on the particular legacy application requirements and the pre-migration analysis results, the migration processes and their flow are explicitly customized. The business model concerns are also studied and defined in this phase. These business issues are included in the application architecture and organizational processes to face the new situation are also (re)defined in this phase. Finally, migration phase includes the verification and validation (V & V) of the final system.
- **Post-migration:** In this phase, the modernized application components will be deployed onto the target environment and it will be checked if both the technical and business objectives established in the pre-migration phase have been achieved. The validation activities that are foreseen are mainly focused on behavioural equivalence, model based testing and end-user functional and non-functional testing. Moreover, a certification model will be created in order to increase customer confidence in the SaaS system.

In addition to the three phases described above, another phase, named **Migration Artefacts Reuse and Evolution**, was defined in order to enable the effective reuse of software artefacts and optimization of the migration processes. This phase includes all needed application maintenance activities after migration to the Cloud (software updates, cloud provider changes, etc.).

The complete methodology is formally described using the Eclipse Process Framework EPF [7] and SPEM2.0 [17] specification and the generated diagrams of each phase are presented in the following sections.

3.2. Pre-Migration Phase. The pre-migration phase (Figure 3.1) is the starting point of each migration. Migration of legacy applications is considered as a very challenging project that involves not only changing the way companies are going to deliver their software, but also their business model, and how the company is organized in terms of processes. Thus, software vendors need to analyse whether the targeted objectives are actually feasible to them both technologically and economically.

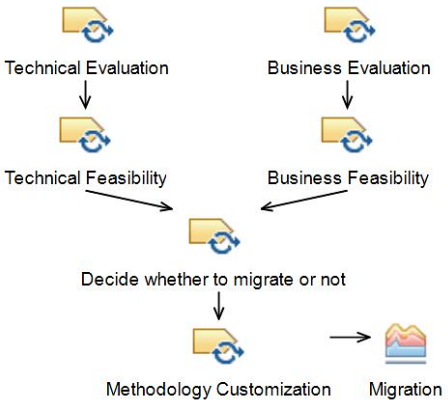

FIG. 3.1. *Pre-migration EPF Model*

The first step in this pre-migration phase is to analyse how mature the application is in terms of technology (i.e. architecture, programming language, database, integration with third party offerings, etc.) and business (i.e. current business model, maintenance and upgrades procedures, etc.). It also considers how the customer wants the application to be in those two axes (i.e. architectural design, cloud provider requirements, business model, legal concerns, performance thresholds values, etc.) once the application is migrated. The analysis of

both the current and ideal situations allows the users to perform a gap analysis, described in terms of a technical feasibility analysis and a business feasibility analysis.

The technical feasibility analysis is aimed to provide a snapshot of the application's design quality, of its complexity and coupling, etc. This is realized thanks to reverse engineering techniques through which the component model of the source code is obtained and an evaluation of the effort required to perform the migration is implemented. The technical feasibility analysis provides a set of migration strategies for each of the components identified in the legacy application and the effort required to perform that strategy. This information is obtained by the combination of the the ideal technological maturity identified in the maturity assessment as well as the target platform requirements, the complexity due to the migration strategy nature and the complexity inherent to source code (the latter is calculated performing a static analysis of the source code).

Furthermore, based on the results from the ideal situation identified in the maturity assessment and the identification of the target platform expected characteristics, a business feasibility analysis is performed. This business feasibility analysis aims at providing not only economic information (ROI, payback, etc.) but also what are the main risks to be faced with the migration and the organizational processes affected by the uptake of the new business model. The results obtained will guide decision makers to identify the most appropriate strategy in terms of migration and selection of the target services/platforms to use.

3.3. Migration and Modernization Phase. The Migration phase is the core part of our methodology, incorporating unique features for reverse and forwards engineering as well as for the effective target environment specification.

3.3.1. Application Discovery and Understanding. During the technical feasibility assessment as well as during the migration process itself, the discovery of relevant models describing the software system is first required in order to 1) have a better understanding of it and 2) provision the other steps of the process with the needed models. Thus, one of the main goals of Model Driven Reverse Engineering (MDRE) is to extract the overall structure and logic of the software system (as well as some more implementation details when necessary for the actual migration), considering different abstraction levels depending on the targeted stakeholders and migration scenarios. In any case, such a MDRE process is generally composed of the two main tasks (Figure 3.2).

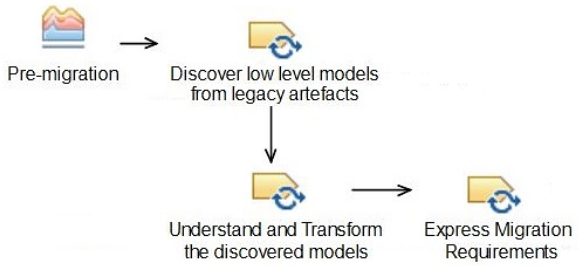

FIG. 3.2. *Application Discovery and Understanding EPF Model*

Model Discovery generates the minimum set of required initial "raw" (i.e. low-level) models out of (some of) the artefacts composing the software system. This is realized (at least semi-)automatically thanks to dedicated software components called model discoverers. This first task notably implies analysis the different available software artefacts to identify the ones which are actually usable and relevant to the considered migration scenario, and also to guide the discovery process itself. This preliminary action can be performed with the help of the *Taxonomy of Legacy Artefacts*, as established and developed within the time-frame of the ARTIST Project, which allows for better classification of these artefacts according to several common dimensions (their technical space, internal structure, nature, size, environment, etc).

Model Understanding produces the necessary "processed" derived models from the initially discovered models, thus filtering only the information required for the remainder of the overall process (i.e. Modernization

notably). This is realized in order to identify and build higher-level views on the analysed software system. This second task relies on the use of various (chains of) model-to-model transformations from the previously obtained "raw" (low-level) models to the final derived models to be provided as useful views and/or inputs of the Modernization tasks (notably). These derived models may conform to different metamodels and so offer several "viewpoints" on the software system at different levels of abstraction, according to the actual requirements of the next tasks and corresponding architects/engineers.

3.3.2. Target Environment Specification. In the Target Environment Specification phase (Figure 3.3) we have two independent processes taking place, one in the application domain and one in the candidate target environment domains, that can be linked at the end for added value. In the first place, Target Environment Benchmarking is performed in order to acquire measurements of cloud services performance in a variety of different application types (e.g. DBs, server based apps, Java based apps etc.). This information is then included in concrete provider models that contain also numerous functional characteristics. This is a process that is performed "offline" and by a suitable role (e.g. Benchmarks provider). Furthermore, in these models we include also cost information, that can be used together with performance aspects, in order to rank services based on their combined metrics, supporting also different ratings based on user interests that are expressed through percentages (e.g. 70% interest on performance, 30% on cost). More information on this process can be found in [8].

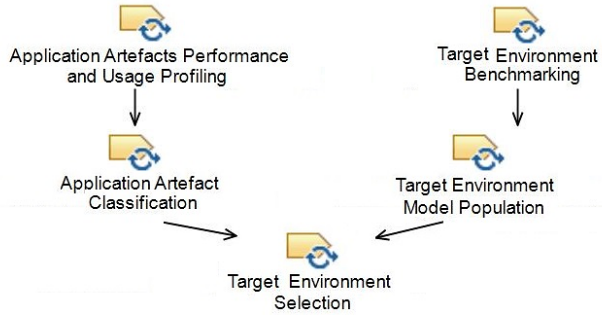

FIG. 3.3. *Target Environment Specification EPF Model*

The second branch of the process is related to the analysis of the performance footprint of arbitrary application components. In this process, which is performed by the Application Developer through the use of the respective tools, the application component's runtime behaviour is compared and classified based on the behaviour of the benchmarks used in the Target Environment Benchmarking, when both are executed on the same reference platform (of the Application Developer). In this process, the developer utilizes the *Benchmarking Tool* (to install and execute the benchmarks locally) and the *Profiling Tool* (to acquire the execution footprint of both the benchmarks and the application component). The final step is to feed this information in the *Classification Tool*, that will perform the matchmaking. Thus, after having this classification conclusion, we can select the cloud service that has the best score in the same benchmark category that the application component has been classified. It is necessary to stress that this part of the process is optional for the developer. If they have already knowledge of their application behaviour or nature, they can directly ask a question of the form "Give me the best offering for running my (web server) application", along with the aforementioned percentages of interest described in the first phase.

3.3.3. Modernization. During the migration phase, once the legacy system has been well understood and decomposed into features and/or components, other tasks such as Model Driven Forward Engineering (MDFE) need to be applied aiming at transforming the legacy system and deploying the migrated one into the target Cloud. During the understanding phase, some high abstract level model views of the legacy system have been produced. These views represent, e.g. the platform independent models (PIM) of the legacy system. Feature or component views are useful since they enable a feature-driven iterative migration across a number of selected features or components. For instance, persistence could be a feature to iterate on: during this iteration all data

sources marked for migration are transformed.

In this modernization phase (Figure 3.4), some artefacts produced during early phases are taken as input: a) the PIM models of legacy system, b) the architectural constraints (e.g. migration goals) corresponding to the target maturity level the application and c) the models describing existing target platforms where the application can be deployed.

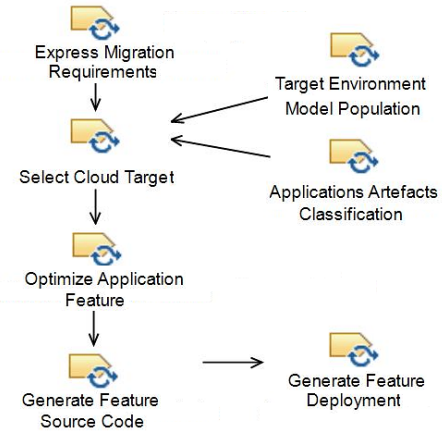

FIG. 3.4. *Modernization EPF Model*

Models of the candidate cloud targets environments are matched against the aforementioned requirements, using a model-enabled matchmaking algorithm. Models describing features or components tagged for migration are optimized and transformed, by applying suitable transformation patterns, into target models. These models represent equivalent features or components that are compatible with the selected target provider and that fully exploit the target features and services requested by the application requirements.

These migration tasks aim at building and deploying the migrated component corresponding to selected legacy components (e.g. implementations of application features). MDFE strongly relies on models and model transformations e.g. model to model (M2M) and model to text (M2T) for the technical materialization of the described MDFE tasks. The main aim for these transformations is to generate diverse models providing different views over the migrated application, including its source code. The skeleton of the application is generated automatically, and developers have to write manually some code to complete upgraded and newly added functionalities.

In cloud applications, the **business model** is tightly intertwined to the technical solution and the methodology defines several tasks in order to define the business model, taking as baseline the principles by Osterwalder [20]. Delivering a product is not the same as delivering a service in terms of organizational processes. Existing processes need to be analysed and redefined to adapt the organization to the new structure. Also, new processes related to the service delivery will have to be defined from scratch and to this end we have identified several processes that are affected by the shift of business models. These are: *Development Process, Updating Process, SLA Management, Helpdesk, Incidence Management, Marketing, Accountability, Cloud Provider and Roles Alignment.*

3.4. Post-Migration Phase. The quality of the migrated system needs to be verified, considering both behavioural (functional) and non-behavioural concerns such as performance or security. The migrated system has to operate similarly to the legacy system and needs to perform at least equally to the old system in terms of functionality and performance. The non-compliance of any of these requirements may cause the non-acceptance of the migrated system by the client. For this reason a validation and certification model is being used as part of the post-migration phase (Figure 3.5).

Once a feature or component has been migrated, its behavioural equivalence (across the legacy and migrated components) and the fulfilment of migration goals (i.e. non-functional requirements) need to be asserted. The first step of this validation covers the functional requirements of the system and is being performed against

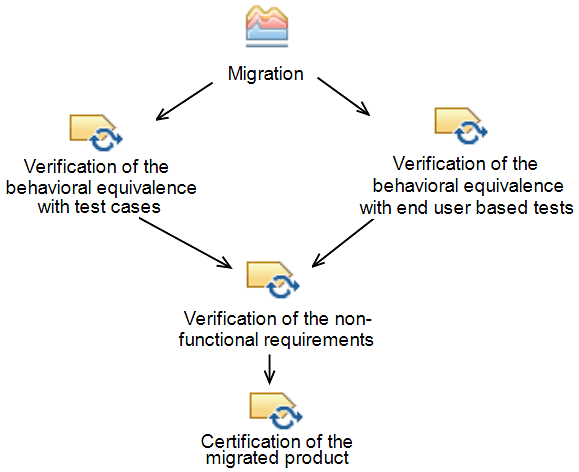

FIG. 3.5. *Post-migration EPF Model*

given test cases as well as provided by the end user test cases. The following step is the validation of the non-functional requirements, in terms of scalability, security, etc.

If the assessment fails, the process rolls back to the migration phase, in order to re-evaluate the migration requirements and the optimization/transformation strategies. Otherwise, the process moves to the next phase with concerns to the current feature. This phase iterates until all features have been migrated. If the assessment is successful, a certificate for the migrated project is being produced and following that the migrated artefact is being placed into the *ARTIST Repository* (explained in detail in the following section).

3.5. Migration Artefacts Reuse and Evolution Phase. The purpose of this supporting phase (Figure 3.6) is twofold: On the one hand, it provides tasks to foster reuse of migration artefacts inside and across migration projects, on the other hand it supports evolving the migrated system after a migration has been performed.

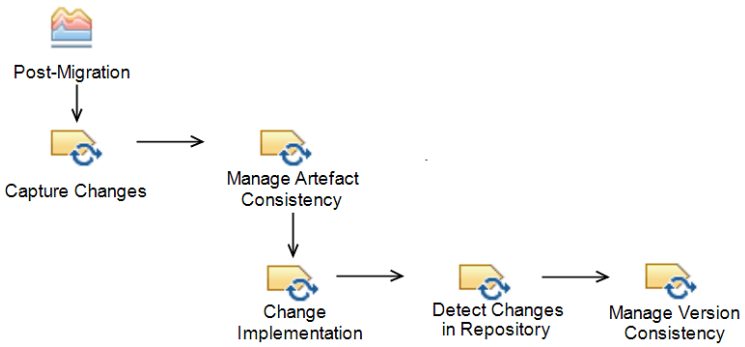

FIG. 3.6. *Artefacts Reuse and Evolution EPF Model*

During a migration project, a number of artefacts that are potentially reusable across projects are produced, such as meta-models, models, generic transformations and other information produced by the various framework tools. By reusing previous results, expensive operations such as big model extraction runs have to be performed only once. The artefacts are first checked for their reuse potential. Reusable artefacts are then published to the web-based *ARTIST Marketplace* if they should be publicly available or to the ARTIST Repository if they should be shared only within a controlled number of migration projects. The ARTIST Repository and Marketplace thus provide a central place to store, archive and organize MDE artefacts which could be merchandized as services in cloud marketplace environments [13].

The second purpose of this phase is to support maintenance activities during and after migration. This is achieved on the workspace level by supplying developers with information about the (modelling) artefacts in the workspace including their types, their relationships and dependencies and their history. This information is gathered from the workspace and from associated source code management and ticketing systems. One possible way to exploit this data is to provide traceability between artefacts and perform change notifications and change impact analyses.

On the artefact level, evolution is supported by a change description service. It can be triggered whenever new versions of one or more artefacts are produced. The service captures changes, detects inconsistencies in depending artefacts and supports the user in resolving these inconsistencies. Finally, the initial and the derived changes need to be implemented to complete the change workflow.

4. Architecture.

4.1. Architecture Overview. The architecture design is an integrated view of the functional blocks and components composing the ARTIST tooling, stereotyped with the main methodology block, i.e. pre-migration, migration and post-migration, to introduce the various components in turn.

The Migration Feasibility Assessment tooling comprises the Maturity Assessment Tool, the Business Feasibility Tool and the Technical Feasibility Tool. These tools interact with each other iteratively to estimate and report about the business and technical migration feasibility. This report is used by the Methodology Process Tool (described in next section) to tailor the particular migration process. The Migration Feasibility Assessment tools rely on some of the MDRE features provided by the Model Discovery and Understanding tools, offering low and high level abstraction platform-specific and independent models (PSM, PIM) of the legacy application.

The Cloud Metamodel and the model instances, describing target cloud providers and offerings, are provided by the Target Environment Specification tooling which comprises the Benchmarking, Performance Stereotype Classification and Profiling tools. These (meta)model artefacts are used during the modernization assessment (e.g. to specify cloud target and migration requirements) but also during the migration phase by the MDFE tooling (e.g. to specify requirements on the "cloudified" application and the cloud target environment). The MDFE Migration tooling comprises Target Specification, Optimization and Deployment tools, which supports the entire migration phase, such as the specification of requirements for the application and the target environment, target lookup and selection, application optimization (e.g. "cloudification") and application building and deployment.

The post-migration phase (Testing, Verification and Certification) is supported by a suite of Testing Tools and the SbSp (Service based Software providers) Certification Tool. Finally, the ARTIST Marketplace and ARTIST Repository realize the migration artefacts reuse and evolution phases and operate complementary to the other components so as to improve the effectiveness of various processes that these components perform.

4.2. ARTIST Suite. Following the design of the architecture, a concrete implementation approach for the ARTIST Suite, focusing on the support for the migration of both Java and .NET based applications is described in Figure 4.2.

The ARTIST suite comprises these main blocks of tools:

- **Browser-based (Web-based) ARTIST Suite**, targeting mostly end-users playing a **business role**. This tooling is intended to offer functionalities that require broader access to different types of users, compared to the more technical tooling which requires more specialized tools. Examples of tools in this block are: the Maturity Assessment Tool or the SbSp Certification Tool.
- **Eclipse RCP based ARTIST Suite**, targeting mostly end-users playing a **technical role**. Both the Eclipse platform itself and the Eclipse Modelling Project (EMP) tools, as baseline for ARTIST suite, are the natural choice looking to the selection of MDE techniques and OSS tools. Examples are those addressing technical functionalities, such as the Technical Feasibility Tool, the Model Discovery and Understanding, the Requirements Specification Tool, the Optimization Tool, the Target Generation Tool, the Deployment Tool or the different Testing Tools.
- **Sparx Enterprise Architect (EA)** [23], that includes a set of add-ons developed which support the discovery and target generation of .NET C# based applications.

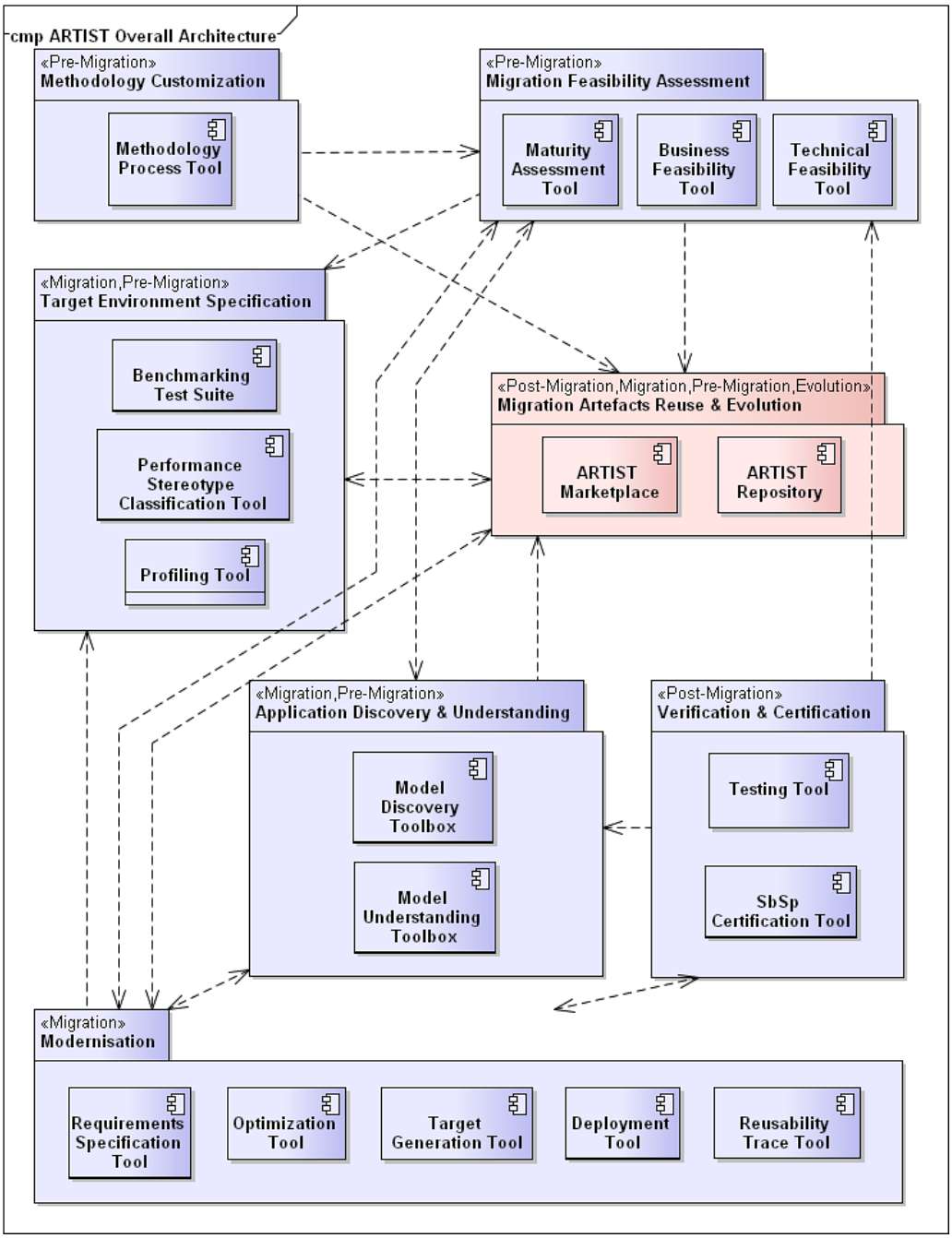

FIG. 4.1. *Overall Architecture Design*

The proposed suite supports the migration of both Java and .NET applications. However, Eclipse support for .NET development is somehow limited. .NET applications can be developed using Microsoft Visual Studio or other corresponding development environments. Our approach for the migration of .NET application relies on the usage of Sparx EA, during certain tasks of the methodology (notably during the Model Discovery and the Target Code Generation tasks). Thus, models of .NET applications will be obtained out of .NET projects using EA, and the final target code will be generated out of the migrated models using EA as well.

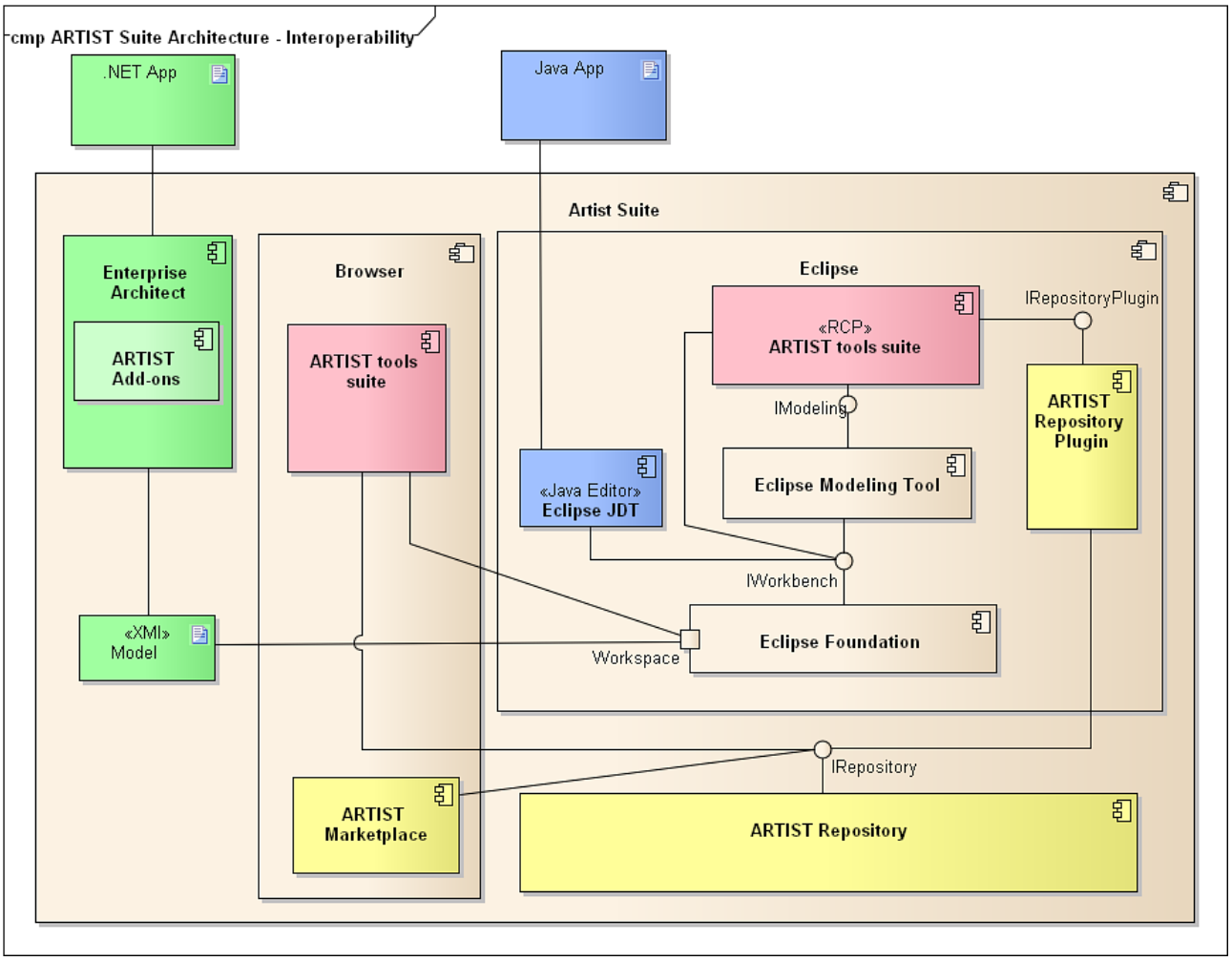

Fig. 4.2. *ARTIST Suite Architecture - Interoperability*

The remainder of the methodology tasks involving modelling are supported by Eclipse ARTIST tooling. The migrated .NET code can be compiled and assembled for deployment using Microsoft Visual Studio or another compatible development environment. Similarly, Java-based applications can be developed using the Eclipse JDT or another Java development environment. The projects of these Java applications can be then easily imported into the Eclipse workspace, ready to be managed by the tools. EA and Eclipse ARTIST tooling exchange models (notably UML ones) using the XML Metadata Interchange (XMI) [18] serialization format.

We foresee that the interactions between the browser-based and the Eclipse-based suites will be user driven and mediated through the Eclipse workspace and the repository. The user will save, locally on his/her workspace or in the repository, the artefacts produced during the performing of concrete tasks of the methodology and created using the browser-based tools. In turn, the user will import them within the Eclipse-based tools when required. The marketplace also offers browsing and searching functionalities to the end-users. Thus, the marketplace is the access point for users to the repository. Nevertheless, the various tools can access the repository directly through the repository plugin to retrieve/save artefacts provided that they have got the artefacts URIs.

Eclipse-based suite is tightly integrated within the Eclipse framework and the Eclipse Modelling Project (EMP) tools, using the Eclipse platform API and thus contributing to the Eclipse workbench. Artefacts produced and consumed by the Eclipse-based suite will be shared with other tool suite instances through the

repository by using the repository plugin. Alternatively, these artefacts can be stored locally in the Eclipse workspace for personal usage.

5. ARTIST Methodology Process Tool Implementation. The methodology is a modernization and migration solution that covers a wide range of application types, regardless of the underlying technologies, business models, operational modes, etc. In order to practically support this methodology, a core element was incorporated in the overall architecture: the Methodology Process Tool (MPT). This tool allows the customization and instantiation of the methodology based on the specific application requirements.

The Methodology Process Tool, exploiting the results processed and obtained during the modernization assessment, defines a customized modernization process, tailored to the concrete legacy application needs. The tool shows the customized process in detail, its tasks broken down step-by-step, including hooks to invoke the tools required to accomplish each task.

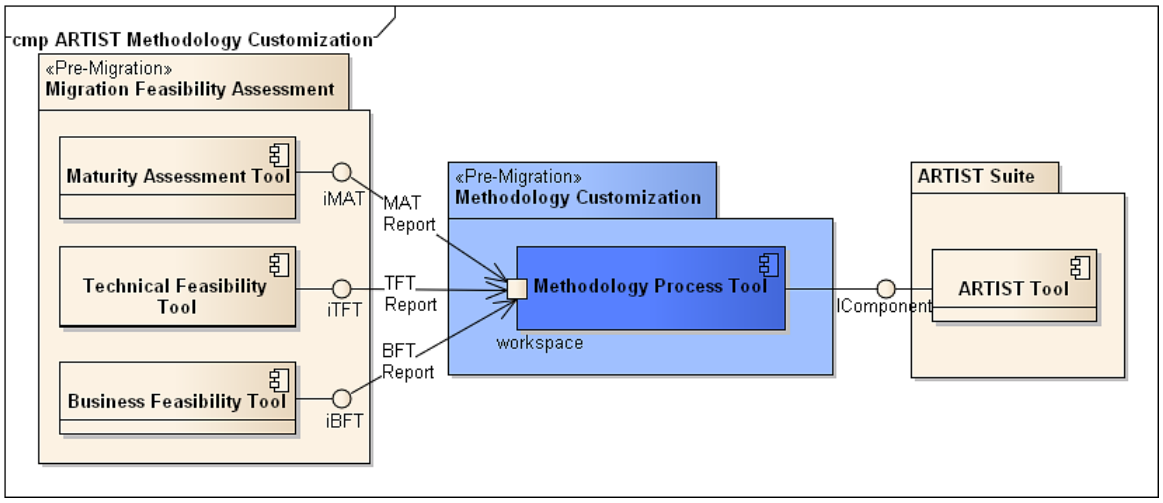

FIG. 5.1. *ARTIST Methodology Process Tool Dependencies*

MPT is the main component of the framework supporting the customization of the methodology at the end of the pre-migration phase. The methodology components are those which, based on the results of the modernization assessment (including the target specification), provide a tailored modernization blueprint. At the end of the pre-migration phase, after a positive early assessment (e.g. migration accepted), the remaining migration process (e.g. the rest of phases for completing the migration process) can be tailored to the specific characteristics of the migration project (i.e. legacy application and selected migration goals) using the Methodology Process Tool (MPT), which creates a specific blueprint for the migration project, that is, a specialization of the methodology process, rendered as a graphical process, showing tasks as visual elements (e.g. widgets) and related tasks for each phase within a group widget, logically connected through the methodology workflow. Moreover, each task widget includes links to the tools used to accomplish it, therefore selecting a task triggers the corresponding task tool in the integrated Tool Suite (e.g. Eclipse environment).

MPT uses reports generated during the early assessment phase, since the technical and business feasibility assessment reports, contains the required information to particularize the methodology to this concrete migration project. MPT can programmatically launch any required tool to accomplish any of the tasks described in the tailored methodology blueprint.

With respect to the overall architecture and methodology and the requirements posed by the objectives of the project, several approaches have been examined for the implementation of MPT. Currently we focus on a) a hybrid implementation for MPT, which combines features of a web-based platform and of the Eclipse environment and b) on an implementation using the SpikesTogether platform [24]. Both implementations are discussed in the following sections.

5.1. Hybrid (Web+Eclipse) Implementation. The MPT hybrid tool was built to take advantage of both the Web and the Eclipse based MPT architecture choices and combine them together to support the customization of the methodology from both any web browser (e.g. to support non-technical users, such as business analysts) or the Eclipse-based Tooling (e.g. to support technical users, such as modelers). In this way, the user is able to access MPT functionality either from an external Web browser, which connects to the web server that host the MPT, or within the Eclipse-based tooling, using the internal Eclipse Web Browser. Besides, this also allows the usage of Eclipse cheat sheets [5] to guide the user through the entire customized methodology and launch other Eclipse-based tools, as required by each task of that methodology.

Another challenge behind this implementation was to effectively exploit the representation of the methodology that was implemented using the Eclipse Process Framework, as described in the previous sections. For this reason, the methodology description outcome, which was in the form of simple static HTML pages, was transformed to XTML and integrated with Java Server Faces in order to achieve their dynamic customization according to the specific case under examination.

This implementation of the MPT provides to the end user the ability to upload and share existing reports in terms of maturity assessment, technical and business feasibility analysis, as produced from the corresponding tools, namely the MAT (Maturity Assessment Tool), TFT (Technical Feasibility Tool) and BFT (Business Feasibility Tool) tools respectively (see Figure 5.2).

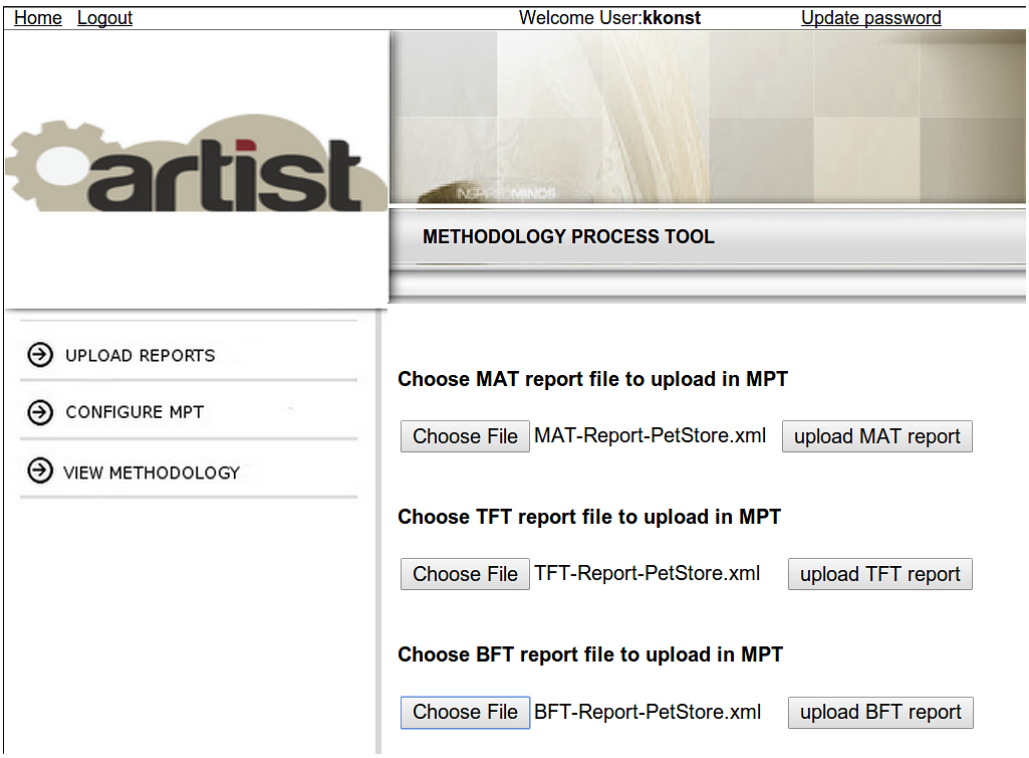

FIG. 5.2. *MPT Hybrid Tool: UPLOAD REPORTS*

Once the reports have been uploaded, the user is able to select among these reports and use the appropriate one to configure the methodology processes, according to the attached results for maturity, technical and business feasibility assessment. It should be noted that it is also possible to access existing reports that have been uploaded by the same user in the past for other projects as well as reports that other users are willing to share (see Figure 5.3).

By using the Methodology Configuration option, the selected report in XML format is being parsed and the extracted values are being stored and used against rules, the outcome of which may affect the content of

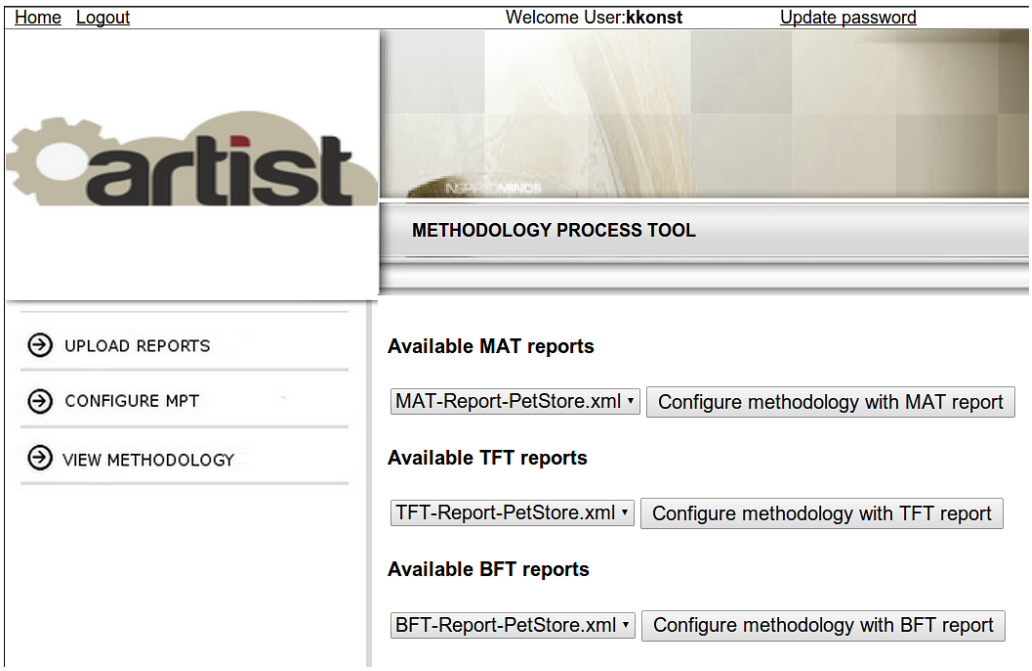

FIG. 5.3. *MPT Hybrid Tool: CONFIGURE MPT*

each methodology element (phase, task, activity) at different levels or even the flow that the user should follow in order to cloudify the project under examination.

Following the configuration of the methodology, the user can view the customized version of the EPF by clicking on the "View Methodology" button on the left pane. This action opens the index page of the EPF outcome in a new tab. An example for the methodology customization is described in the evaluation section.

5.2. SpikesTogether Implementation. A SpikesTogether-based approach has also been implemented focusing on accessibility towards business users. SpikesTogether [24] is developed on top of SharePoint [14], a well known web-based business collaboration platform. SpikesTogether is a tool which facilitates the development of line-of-business (LoB) solutions, extending the core functionality of SharePoint in particular key aspects.

SpikesTogether is built around two cornerstones: a) Adaptive Case Management and b) Collaborative Workflows. In the context of this work, application migration can be seen as a case to be managed while the migration process can be seen as a workflow. An actual application migration to the cloud typically involves many people which are supported by the collaborative aspect in the workflows. In addition a customized migration project is supported by the adaptive aspect in the case management.

Figure 5.4 shows a screenshot of a workflow in progress (i.e. the proposed methodology) on a particular item or migration project (i.e. SpikesTogether Migration). Tasks of the workflow in SpikesTogether are assigned to a user involved in a particular role. Every task has a certain outcome possibly leading to other tasks until the workflow (and consequently the application migration) has been completed. The figure also shows that the current migration is in the Migration phase which has not yet started while the pre-migration phase has already been completed. The example workflow used in this process can be seen in Figure 5.5. This workflow involves three different roles (i.e. Analyst, Modeler and Tester) each performing one particular step of the methodology process (i.e. pre-migration, migration and post-migration).

The SpikesTogether implementation guides the different users (in different roles) throughout the complete migration process and provides them with clear descriptions of the tasks that need to be performed using one or more of the tools in the tool suite. Identity/access management as well as security is being provided by the underlying SharePoint platform. Currently, the focus was around the step-by-step guidance of the user throughout the migration process while future work in this approach includes the customization of the

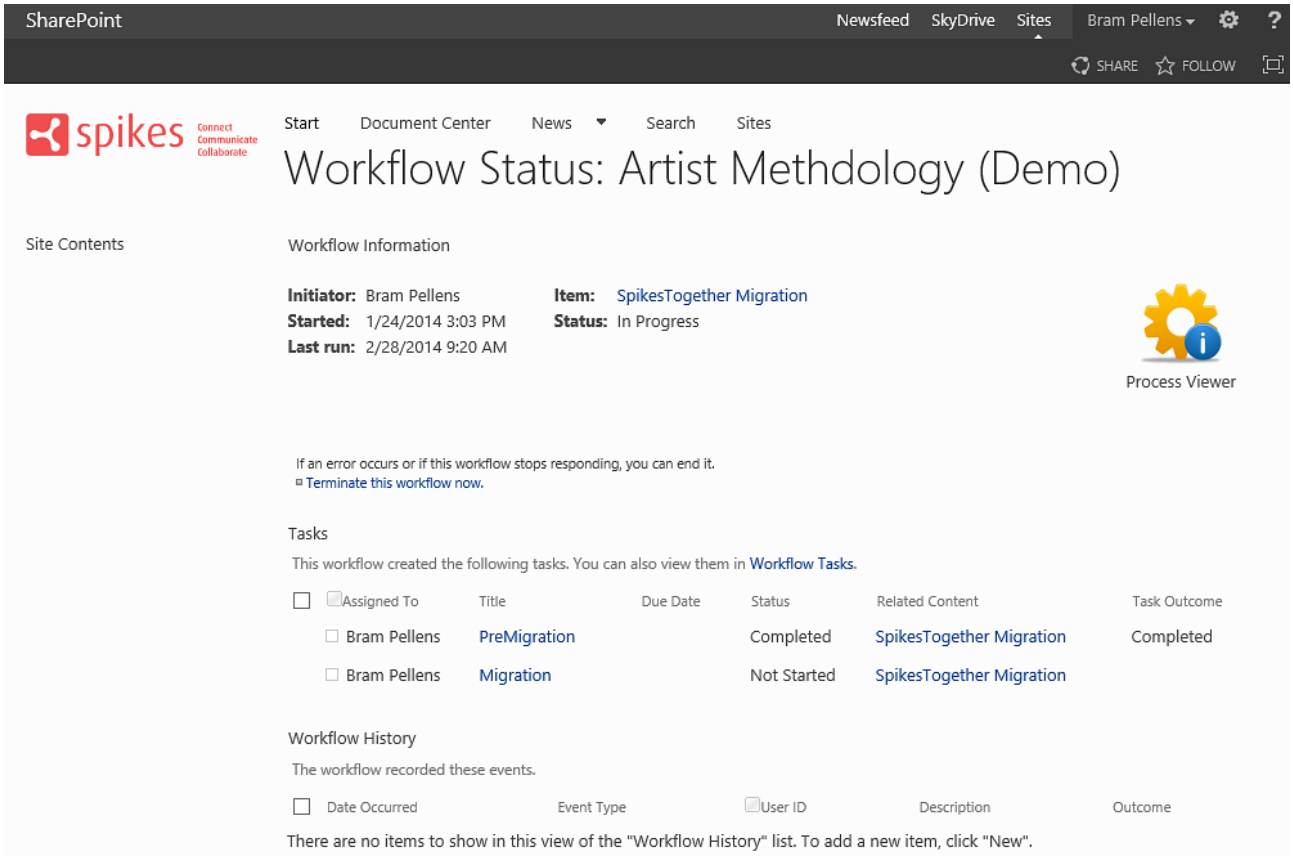

FIG. 5.4. *SpikesTogether MPT Implementation*

methodology (i.e. adaptation of the workflow).

6. Proof of Concept and Evaluation. In the previous sections, we described a generic but also detailed methodology for the migration of legacy software to modern target environments, which is also supported by a tool for its effective customization to the specific requirements of a particular migration project. In this section, the proof of concept for the Methodology Process Tool based on an experimental migration project is discussed. The proof of concept will show how specific steps of the generic methodology are customized for a concrete migration project based on the pre-migration analysis results. The Evaluation of the proposed methodology and its respective implementation, the Methodology Process Tool, was based in an experimental legacy J2EE Application, the PetStore [19]. In the following sections we describe the steps for the business and technical analysis of the application during the pre-migrations phase and then, based on the analysis results, we describe the rule-based customization of the methodology through MPT.

6.1. PetStore Business Case. Our objective is to tackle the problem of software modernization under two perspectives, the technical one and the business one. This is the reason why **we need to define the business case of the PetStore in order to evaluate and test all the tools under both perspectives**:

SME-Software is an English company producing and marketing software products targeting SME Enterprises. Their customers include Hotels, Restaurants, Stores, Chartered Accounting Firms, Employment Law specialists. PetStore application started as a management system or ERP for stores selling pets, and evolved as a web portal for selling pets over Internet.

Nowadays SME-Software sells its application by licensing (to be renewed each year) and offers the possibility to customize it through consultancy services via ad-hoc projects. This customization includes the purchase of the new infrastructure (Application server, database server, etc.), installation of the adequate application server

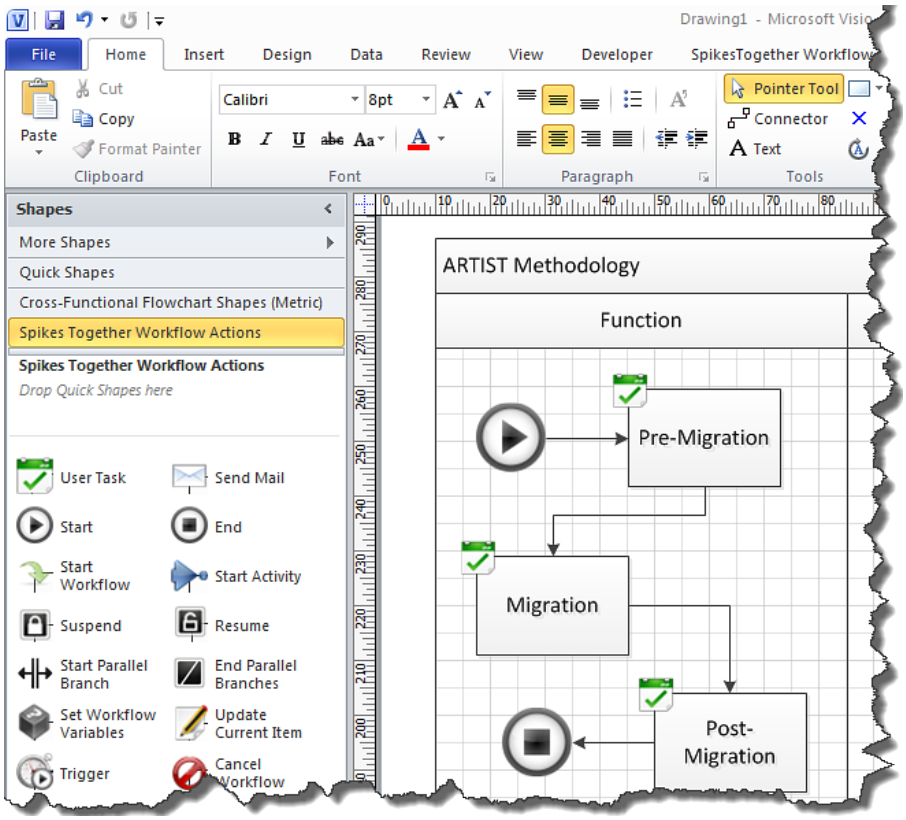

FIG. 5.5. *SpikesTogether MPT Usage Example*

and database server software versions to fit the PetStore requirements, and customization of the interface layer. The customers may contract these services to companies other than SME-Software. Optionally, customers can contract maintenance support.

With the emergence of cloud computing and the SaaS business model, SME-Software managers have start thinking about offering their Software as a Service. Many customers have shown willingness to contract the service instead of buying the product and the infrastructure required to install it.

SME-Software's CIO is aware of the emergence of cloud computing and he has participated in discussions where the advantages of cloud computing have been exposed. He notices that their customers are demanding changes in the way they sell and offer their product but he is still reluctant to change because the following aspects are not known:

- *implications to change from SaaP (Software as a Product) to SaaS*
- *time and costs implied from a potential migration*
- *which IaaS provider is the best, and which implications this could have*
- *which risks (market and technical) are associated to this change*
- *which business processes are affected and how to adapt them to the new selling model.*

6.2. PetStore Software and Business Case Assessment. MPT tool will personalize the methodology for each migration project relying on the results obtained in the pre-migration phase by the different tools (MAT, TFT and BFT). From MAT, the migration goals and high level recommendations will be used, from TFT, the list of migration tasks and affected components and finally, from BFT the selected business scenario. By combining all this information, MPT will personalize the generic methodology for each specific migration project.

6.2.1. MAT results. The Maturity Assessment Tool results on application positioning with respect to its cloud maturity are the following:

- **Migration Goals**
  - Migrated programming language: J2EE
  - Multitenancy level: Multitenant
  - Database scalability: High scalability of data
  - Interoperability requirements: Yes, ERP
  - SLA requirements: System availability, performance and QoS, Outages covered, Data integrity, business continuity
  - Configuration requirements: User personalization
  - Authorization requirements: ID, Password
  - Cloud provider: Amazon, Public Cloud
  - Elasticity: Component / self-made means to ensure its elasticity
- **High level recommendations**
  - Model the application to have a better knowledge of the application
  - Redesign the database to be multitenant
  - Redesign the architecture and distribution of stateful/stateless nodes
  - Integrate a module of concurrent users monitorization
  - Define authentication mechanisms
  - The coexistence of the two business models may bring difficulties in the support processes
  - Define new SLAs taking into account the infrastructure part

6.2.2. TFT results. The Technical Feasibility Tool is aiming at estimating the efforts required to migrate a legacy application to a selected target cloud environment, fulfilling some migration goals and requirements. This effort estimation is broken down into each legacy component and the migration tasks required migrating it. The effort required to accomplish each migration task is derived from the complexity of the task and the complexity inherent to the legacy code (component complexity). TFT analysis results are summarized in the following table. In the example shown, the different components of the PetStore have been identified ("component" column) and the following metrics have been obtained for each of them:

- **Component complexity**: This is the complexity related to the legacy software. In this proof of concept, just for the sake of simplicity, component complexity is just measured by the number of files comprising the component: 1 XML file for J2EE resources (complexity 1.0) and 40 Java classes for PetStore (complexity 40.0), 1 SQL file for PetStore data schema, 1 bundle file (installation) for J2EE Server and Non-SQL Server. More elaborated complexity based on the Maintainability metric and SW/OOP/Coupling metrics will be computed by TFT.
- **Task**: This is the migration task suggested by TFT (following a rule based approach) to migrate the corresponding component.
- **Task Type**: Each task is been characterized by its type. Task taxonomy (including Installation and configuration tasks, Data sources related tasks, Code refactoring tasks, Connection/Configuration tasks) has been defined in the context of the TFT where the knowledge representation of expert judgement with respect to the complexity of the task is embedded.
- **Task Complexity**: The task complexity is obtained by the TFT based on the selected task complexity level and expert heuristics and/or historical data.
- **Task Effort**: This is the required effort to complete the task. The effort is computed based on the component complexity, task complexity and expert heuristics and/or historical data.

6.2.3. BFT results. In the business feasibility analysis the following business scenarios were identified:

- **PetStore In-house**: This scenario represents the Business Model as-is for SME Software Co. at the time of the assessment. PetStores buy licenses from SME Software Co. PetStores rely on external organizations for maintenance of hardware and software and provisioning of resources (e.g. hardware, energy). Customers buy pets from PetStores using the PetStore Software.
- **PetStore on IaaS**: This scenario presents the Business Model assuming that the PetStores rely on IaaS Providers to operate the PetStore service: PetStores purchase SW License from SME Software Co.

TABLE 6.1
TFT Results

Component		Task			
Name	Compl.	Name	Type	Compl.	Effort
J2EE Server	1.0	App Server Installation & Configuration	Installation and Configuration	2.0	2.0
Non-SQL Server	1.0	Non-SQL persistence framework installation and configuration	Installation and Configuration	2.0	2.0
PetStore Web App	40.0	PetStore Persistence Layer re-coding based for Non-SQL persistence framework	Code Refactoring	5.0	40.0
PetStore Web App	1.0	PetStore data schema refactoring for Non-SQL persistence framework	Data Source	5.0	8.0
Non-SQL Server	1.0	PetStore data dump into Non-SQL persistence framework	Data Source	1.5	1.0
JDBC Resource	1.0	PetStore JDBC Resource reconfiguration	Connection-Configuration	1.0	0.1
Connection Pool	1.0	PetStore connection pool reconfiguration	Connection-Configuration	1.0	0.1
				Total	53.2

In this scenario, the SW is licensed at a lower price. The PetStores rely on IaaS Providers to operate the PetStore service. In turn IaaS purchase resources (e.g. hardware, energy) from external resource providers. Customer buys pets from PetStores using the PetStore Software. PetStores pay royalties to SME Software Co. for each transaction completed by means of the PetStore services

- **PetStore on PaaS without subscription fee**: SME Software integrates the PetStore software relying on the platform of a PaaS Provider. SME Software pays the PaaS provider to operate the PetStore service. The IaaS buy resources (e.g. hardware, power) from external resource providers. Customer buys pets from PetStores using the PetStore Software. PetStores pay royalties to SME Software Co. for each transaction completed by means of the PetStore services.
- **PetStore as Cloud Provider**: PetStore pays a subscription to the PetStore service operated directly by SME Software Co. using in-house infrastructure. SME Software Co. pays for both the maintenance the infrastructure and resources acquisition. PetStore pay royalties to SME Software Co. for each transaction completed by means of the PetStore services.

6.3. Methodology Customization and Instantiation for Proof of Concept. The MPT is developed with a set of rules for the customization of the methodology based of the parameters of each migration scenario. In the proof of concept, we present the following example rules which were applied to the pre-migration analysis results of the PetStore application. These example rules are based on the technical and business migration goals defined by the user and extracted from MAT.

Example Rule #1 (technical level): This rule examines a) the parameter *databasereq* which refers to the database requirements (requirements elicitated by the application owner for the database) and b) the

parameter *targetplat* which refers to the (cloud) target platform selected by the application owner (where the application will be deployed). This rule is triggered with the value *high_escalability* for *databasereq* parameter (high scalability of database is required) and with the value *AEC2* for *targetplat* paremeter (Amazon EC2 Infrastructure).

Listing 1
Example Rule for Customization on Technical Level

```
If (databasereq == high_escalability AND targetplat == AEC2)
 1. EMREQ task has to be personalized as follows:
  a. Annotate PIM with AWS requirements;
  b. Annotate the backend component with the requirement
     stereotype "highly scalable database";
 2. OPTAPPFEA task has to be personalized as follows:
  a. Data schema refactoring and optimization to
     target NoSQL persistence frameworks;
  b. Data dump transformation to fit the data
     into the target NoSQL persistence frameworks;
  c. Persistence layer adaptation based on the
     selected target NoSQL persistence frameworks;
end If
```

Example Rule #2 (business level): This rule examines the parameter *bil_rule* which refers to the billing rule that the application owner intends to apply once the application is migrated. This rule is triggered with the value *use* for this parameter that implies direct use of the application.

Listing 2
Example Rule for Customization on Business Level

```
If (bil_rule == use)
 1. REVE task has to be personalized as follows:
  a. Service price definition based on use;
  b. Monitoring and billing component to be incorporated:
     Link to EMREQ and OPTAPPFEA technical tasks;
 2. EMREQ task has to be personalized as follows:
  a. Annotate PSM with monitoring and billing component;
 3. OPTAPPFEA task has to be personalized as follows:
  a. Create the monitoring component at PSM level;
end If
```

Based on the pre-migration analysis and the rules, the methodology is customized so as to define the exact steps that should be followed to accomplish the EMREQ (Express Migration Requirements) task for the specific migration project. Figures 6.1 and 6.2, show the output of two different customizations that affect the way the migration requirements are expressed, covered by the EMREQ task.

Figure 6.1 shows the default view of the page that connects to the EMREQ task. This version appears when the user has not customized the methodology following the process described earlier, or when the reports used for the customization do not trigger a rule that affects the specific task. On the other hand, Figure 6.2 shows a customized version of the same web page in which a rule triggered the addition of extra steps that inform the user that the platform independent model and the backend component should be annotated with specific requirements. In a similar way, all methodology tasks can be customized to the requirements of the particular project so as to ease the navigation through the various tasks and activities simplifying considerably the migration workflow.

7. Conclusions. In this paper we presented a novel software migration approach, which is capable to empower the technical and business capabilities of legacy applications by its modernization and migration to modern cloud environments. The ARTIST Migration Methodology and Framework, as an "all-in-one" solution allows the legacy applications to exploit the offerings of the cloud providers, enrich their unique characteristics and benefit from offering them as services in a potentially global market.

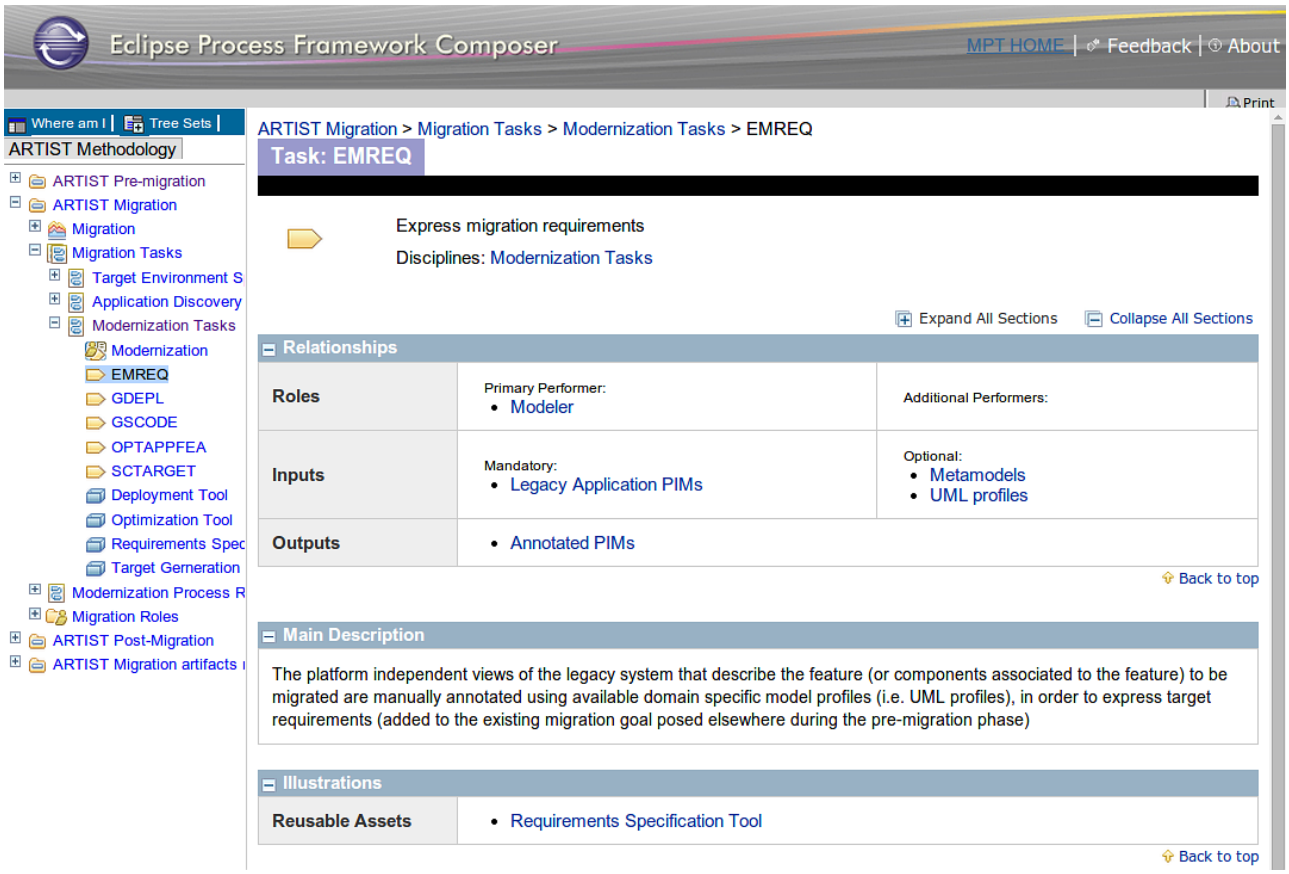

FIG. 6.1. *MPT hybrid tool: Default view of the EMREQ task page (no active rule)*

The added value of the proposed methodology can be summarized as follows: 1) it includes a feasibility analysis before any investment is actually made, 2) it is focused on cloud-compliant architectural issues at both application and infrastructure levels, 3) it includes business model issues that are strongly linked to the technical decisions that are made, 4) it takes into account the impact of the business model shift on the organization processes, 5) it fosters re-usability and automation, 6) it globally prepares the software for its evolution.

The methodology, covering both the technical and business aspects of the migration process, is tailored to the needs of the specific application and guides the owners and developers in every step so as to take full advantage of their software in an effective and productive manner. In addition, a complete set of tools is supporting each methodology task realizing modern analysis and model-driven engineering approaches, allowing when possible the reuse of artefacts. The core of this rich software suite is the Methodology Process Tool which is in charge of the customization and instantiation of the generic methodology based on the pre-migration analysis results of each migration project and the customization rules. The Methodology Process Tool is currently available in two implementations so as to integrate smoothly to the development workflow and preferences of both the JAVA and .NET development communities while further extensions are foreseen to further improve its capabilities and effectiveness.

Acknowledgments. This work has been supported by the ARTIST Project and has been partly funded by the European Commission under the Seventh (FP7 - 2007-2013) Framework Programme for Research and Technological Development, grant no. 317859.

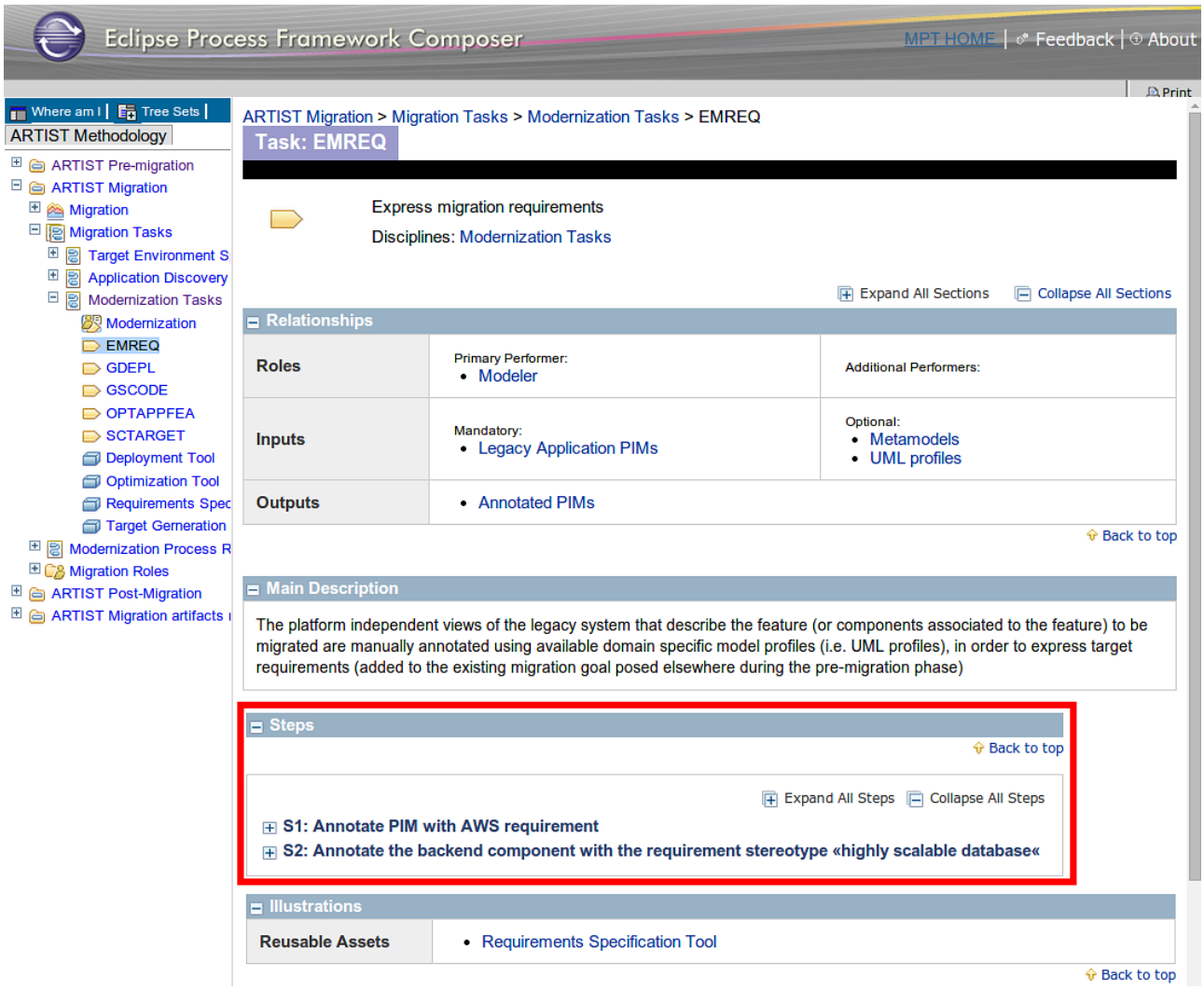

FIG. 6.2. *MPT hybrid tool: Customized view of the EMREQ task page (technical level rule active)*

REFERENCES

[1] ARTIST PROJECT. Available online at: `http://www.artist-project.eu`.

[2] A. BAGNATO, M. SERINA, AND M. MARCO, *In the MOMOCS Tool Suite: a XIRUP Transformation Tool to achieve complex system modernization*, Proceedings OF Model-Driven Modernization OF Software Systems 2008, (2008), p. 101.

[3] H. BRUNELIERE, J. CABOT, F. JOUAULT, AND F. MADIOT, *Modisco: a generic and extensible framework for model driven reverse engineering*, in Proceedings of the IEEE/ACM international conference on Automated software engineering, ACM, 2010, pp. 173–174.

[4] R. BUYYA, C. S. YEO, S. VENUGOPAL, J. BROBERG, AND I. BRANDIC, *Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility*, Future Generation computer systems, 25 (2009), pp. 599–616.

[5] ECLIPSE FOUNDATION, *Eclipse Cheat Sheets*. Available online at: `http://www.eclipse.org/pde/pde-ui/articles/cheat_sheet_dev_workflow`.

[6] ECLIPSE FOUNDATION, *Eclipse Modeling Project*. Available online at: `http://www.eclipse.org/modeling`.

[7] ECLIPSE FOUNDATION, *Enterprise Process Framework - EPF*. Available online at: `http://www.eclipse.org/epf`.

[8] G. KOUSIOURIS, G. GIAMMATTEO, A. EVANGELINOU, N. GALANTE, E. KEVANI, C. STAMPOLTAS, A. MENYCHTAS, A. KOPANELI, K. RAMASAMY BALRAJ, D. KYRIAZIS, T. VARVARIGOU, P. STUER, AND L. ORUE-ECHEVARRIA, *A multi-cloud framework for measuring and describing performance aspects of cloud services across different application types*, in 4th International Conference on Cloud Computing and Services Science (CLOSER 2014), 2014.

[9] G. KOUSIOURIS, D. KYRIAZIS, A. MENYCHTAS, AND T. VARVARIGOU, *Legacy applications on the cloud: Challenges and enablers*

focusing on application performance analysis and providers characteristics, in Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on, vol. 2, IEEE, 2012, pp. 603–608.

[10] G. Lewis, E. Morris, and D. Smith, *Service-oriented migration and reuse technique (smart)*, in Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on, IEEE, 2005, pp. 222–229.

[11] P. Mell and T. Grance, *The NIST definition of cloud computing*, National Institute of Standards and Technology, 53 (2009), p. 50.

[12] A. Menychtas, C. Santzaridou, G. Kousiouris, T. Varvarigou, L. Orue-Echevarria, J. Alonso, J. Gorronogoitia, H. Brunelière, O. Strauss, T. Senkova, et al., *ARTIST Methodology and Framework: A novel approach for the migration of legacy software on the Cloud*, in 2nd Workshop on Management of resources and services In Cloud And Sky computing (MICAS 2013), 2013.

[13] A. Menychtas, J. Vogel, A. Giessmann, A. Gatzioura, S. G. Gomez, V. Moulos, F. Junker, M. Mller, D. Kyriazis, K. Stanoevska-Slabeva, and T. Varvarigou, *4caast marketplace: An advanced business environment for trading cloud services*, Future Generation Computer Systems, (2014)

[14] Microsoft, *Microsoft SharePoint*. Available online at: `http://office.microsoft.com/en-us/sharepoint`.

[15] P. Mohagheghi and T. Sæther, *Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the remics project*, in Services (SERVICES), 2011 IEEE World Congress on, IEEE, 2011, pp. 507–514.

[16] Open Management Group, *Architecture-Driven Modernization (ADM)*. Available online at: `http://adm.omg.org`.

[17] Open Management Group, *Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0*. Available online at: `http://www.omg.org/spec/SPEM/2.0`.

[18] Open Management Group, *XML Metadata Interchange - XMI*. Available online at: `http://www.omg.org/spec/XMI`.

[19] Oracle, *Java Pet Store J2EE Experimental Application*. Available online at: `http://www.oracle.com/technetwork/articles/javaee/petstore-137013.html`.

[20] A. Osterwalder and Y. Pigneur, *Business model generation: a handbook for visionaries, game changers, and challengers*, John Wiley & Sons, 2010.

[21] S. Rugaber and K. Stirewalt, *Model-driven reverse engineering*, Software, IEEE, 21 (2004), pp. 45–53.

[22] D. C. Schmidt, *Model-driven engineering*, COMPUTER-IEEE COMPUTER SOCIETY-, 39 (2006), p. 25.

[23] Sparx Systems, *Enterprise Architect*. Available online at: `http://www.sparxsystems.com/products/ea`.

[24] Spikes, *Spikestogether*. Available online at: `http://www.spikestogether.com`.

[25] I. Warren and J. Ransom, *Renaissance: a method to support software system evolution*, in Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International, IEEE, 2002, pp. 415–420.

[26] B. Wu, D. Lawless, J. Bisbal, J. Grimson, V. Wade, D. O'Sullivan, and R. Richardson, *Legacy systems migration-a method and its tool-kit framework*, in Software Engineering Conference, 1997. Asia Pacific... and International Computer Science Conference 1997. APSEC'97 and ICSC'97. Proceedings, IEEE, 1997, pp. 312–320.

Edited by: Florin Fortiş
Received: Mar 1, 2014
Accepted: Jun 24, 2014

DOI 10.12694/scpe.v15i2.981
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 2, pp. 153–168. http://www.scpe.org

ADAPTIVE TIME-BASED COORDINATED CHECKPOINTING FOR CLOUD COMPUTING WORKFLOWS

BAKHTA MEROUFEL[1] AND GHALEM BELALEM[2]

Abstract. Cloud computing is a new benchmark towards enterprise application development that can facilitate the execution of workflows in business process management system. The workflow technology can manage the business processes efficiently satisfying the requirements of modern enterprises. Besides the scheduling, the fault tolerance is a very important issue in the workflow management. In this paper, we analyse and compare between some existing checkpointing strategies, then we propose a lightweight checkpointing adequate to the cloud computing and the workflows characteristics. The proposed strategy is an Adaptive Time based Coordinated Checkpointing *ATCCp*, it ensures a strong consistency without any synchronization. *ATCCp* uses the concept of soft checkpointing to minimize the storage time and it uses the *VIOLIN* topology to improve the checkpointing performances. According to the experimental results, our approach decreases the overhead and the SLA violations.

Key words: Cloud computing, Workflow, Fault tolerance, Virtual machines, Checkpointing, Coordination, *VIOLIN*, Consistency state, Stable memory, Overhead.

AMS subject classifications. 68M14, 68M15, 68M20, 68W15, 94C12

1. Introduction.

The cloud provisions pools of computing resources as services via the internet using a pay-as-you-go price model that eliminates initial costly capital investments in hardware and infrastructure. Research and academic communities can leverage the benefit of the cloud price model for their computation-intensive applications that traditionally run in HPC environments [34], [35], [36], [37] such as Amazon Web Services' HPC offering [38] or science cloud initiatives [39].

The HPC cloud market paradigm are usually smaller scale HPC users, such as small companies and research groups who have limited access to supercomputer resources and varying demand over time. The perspective of cloud providers is expanding their offerings to cover the aggregate of these HPC. The user jobs (HPC or not) are presented as a workflows to facilitate the resources and the tasks management.

The workflow is a set of atomic tasks, interconnected in a directed acyclic graph through control flow and data flow dependencies. Moving workflows to a cloud computing environment enables the utilization of various cloud services to facilitate workflow execution. Besides the workflow scheduling, the failures are an important issue to deal with during the management of workflow execution.

To ensure the cloud reliability, the system must contain a fault tolerance module to allow the cloud to continue providing the services despite the occurrence of faults. Several fault tolerance techniques are proposed such as:

- Check pointing/Restart: when a task fails, it is allowed to be restarted from the recently checked pointed state rather than from the beginning [23], [12].
- Replication: it is a process of maintaining different copies of a data item or object. Various task replicas are run on different resources, and the execution is successful if at least one replicated task is not crashed [7], [14], [40]. Replication is a resource demanding.
- Resubmission: whenever a failed task is detected, it is resubmitted either to the same or to a different resource [16].

Replication suffers from large resource consumption and resubmission can significantly delay the overall completion time in case of multiple repeating failures. In this paper we use the fault tolerance based on checkpointing. In a distributed system, since the nodes in the system do not share memory, a global state of the system is defined as a set of local states, one from each node. In this case, transit and orphan messages need to be handled; otherwise they can lead to inconsistent checkpoints [12].

[1]Dept. of Computer Science, Faculty of Exact and Applied Sciences, University of Oran - Es Senia, Oran, Algeria (bakhtasba@gmail.com).
[2]Dept. of Computer Science, Faculty of Exact and Applied Sciences, University of Oran - Es Senia, Oran, Algeria (ghalem1dz@gmail.com).

Orphan messages are messages whose "receive events" are part of the destination process checkpoint but the corresponding "send events" are lost. In case of a recovery, the destination process would receive those messages twice, which could result in unpredictable application behavior. On the other hand, transit messages occur when the "send events" are part of the sender-side checkpoint but the "receiving events" are lost.

Figure 1.1 presents three virtual machines (VMs), the horizontal lines represent execution process of each VM, with time progressing from left to right. An arrow from one VM to another represents a message being sent, with the send event at the base of the arrow and the corresponding receive event at the head of the arrow. Internal events have no arrows associated with them. Given this graphical representation, it is easy to verify the communication among VMs. The black boxes illustrate the checkpointing of each VM. C1 is the recorded global state and it consists of the set of VMs checkpoints.

In the scenario presented in Figure 1.1, the message *m2* is a regular message since its send and receive events are not stored in C1 (*m2* was sent and received after C1). The *m1* is a transit message because it was sent before C1 and received after C1 (the "send event" of *m1* is recorded in the global state C1 but the "receive event" is not). In case of a rollback, the VM_2 will wait the *m1* to be sent from VM_1, but VM_1 has already sent this message (according to C1). The message *m3* is an orphan message because its "receive event" is recorded in C1 but the "send event" in not recorded. In case of rollback, VM_3 receives *m3* twice (one recorded in C1 and other sent by VM_2 after the rollback) which can lead to incorrect and inconsistent computations.

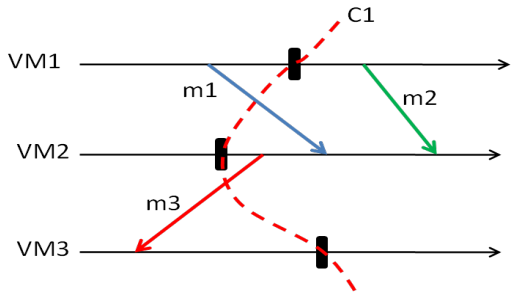

FIG. 1.1. *Communication and checkpointing among three VMs*

The checkpointing must create a consistent state. A state without any orphan message is consistent. If a consistent checkpointing do not create any transit messages then the consistency is strong. To satisfy that consistency, literature offers three main solutions:

- Independent checkpointing (Uncoordinated) [4], [5]: the checkpoints at each process are taken independently without any synchronization among the processes. Because of absence of synchronization, there is no guarantee that a set of local checkpoints taken will be a consistent set of checkpoints. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [4]. The independent checkpointing stores all the checkpoint files during the job life.
- Coordinated or synchronous checkpointing [1], [12]: the processes will synchronize to take checkpoints in such a manner that the resulting global state is consistent. The main advantage is that it stores only one permanent checkpoint in the stable memory and it is domino-effect free.
- Communication induced checkpointing: the processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken independently, while forced checkpoints are taken to guarantee the eventual progress of the recovery line. However, the messages are piggybacked and useless checkpoints can be created.

Even if the checkpointing is widely used in cloud computing environment [18], there are very few papers that study or take into consideration the influence of checkpointing technique. The random choice of checkpointing technique adds large overhead and decreases the system performance considerably. After studying a real execution trace of a cloud computing, it is proved that 80% of communications happen inside the server (among VMs) [8].

In this paper we propose an intra-sever checkpointing strategy. The inter-server can easily managed by using the communication induced checkpointing at servers level [6]. The proposed checkpointing approach is low-

overhead time-based coordinated checkpointing (*ATCCp*). *ATCCp* technique does not need any synchronization and it stores only one checkpointing file in the stable memory which decreases the overhead and the resource consumption. An initiator is selected in *ATCCp* to manage the resynchronization and to solve the timer problems.

The rest of this paper is structured as follows: Section 2 explains some related works. Section 3 presents the adopted system model and its characteristics. Section 4 analysis in detail the time based coordinated checkpointing strategies proposed in literature. In section 5, a description of *ATCCp* is given; a comparative study is also presented in the same section. The experimental results are presented in section 6. The conclusion and future works are presented in section 7.

2. Related works. Besides the scheduling, fault tolerance is a very important issue in workflow management. Literature proposes many works in this field. The authors in [7], [14] use the replication to tolerate the failures. This strategy is very effective when the system uses a deadline for the running application. However, the replication is resource demanding and it can add complexity to the system. The resubmission is also another strategy to ensure the correct execution of the workflow. The authors in [16] balance between the resubmission and the replication to minimize the disadvantages of each strategy. Even with this combination, fault tolerance decreases always the system performance and increases the SLA (service level agreement) violation [1].

The previous strategies (replication and resubmission) are used only for fault tolerance. Checkpointing is a fault tolerance technique but it can be used to ensure or improve other services. In [1], [15], [13] the authors use the checkpointing to ensure a fault tolerant scientific workflows. In this case, the system records periodically its state by checkpointing all the running tasks. The proposed approach in [1] uses two phase blocking coordinated checkpointing which considerably increases the system overhead. The authors in papers [15], [28] use independent checkpointing which causes the domino effect. Independent checkpointing is used also in [4], [5] to improve the migration and ensure the load balancing in cloud servers.

The fault tolerance service proposed in [28] and [29] chooses between replication and checkpointing to tolerate the workflow depending on several criteria such as the cost and the overhead. In [27], a comparison is done between the replication only and the combination between replication and checkpointing. The experimental results prove that the checkpointing improves the replication performance.

It is clear that the checkpointing is a very popular technique and it can be used to ensure many services at the same time. However, random choice of checkpointing technique can increase the system overhead and the SLA violation. According to Section 1, coordinated checkpointing ensures the consistency without the scalability and it is the contrary in case of uncoordinated checkpointing.

Time based coordinated checkpointing is a combination between both strategies (coordinated/ uncoordinated) and it ensures both scalability and consistency with the minimum overhead. Many papers studied and used this type of checkpointing. The majority of the proposed time based checkpointing suffer from many problems and imperfections. Before presenting and analyzing this type of checkpointing, we present in the next section the system model.

3. System model. The cloud computing is a set of datacenters geographically distributed. Each datacenter contains multiple physical machines (servers/Hosts). The hypervisor [9] in each server provides a virtualized hardware environment to support running multiple operating systems concurrently in different virtual machines (VMs) using one physical server (See Figure 3.1). The VM can accept a special user request to execute an application. All the virtual machines can concurrently serve, with different operating systems and software configuration environments according to the need of the user. The kernel level in the VM contains the communication and the task management modules.

We have extended this architecture by implementing a checkpointing module (CP module) inside each VM. We assume also that each datacenter contains a stable storage memory where checkpoint files will be stored. All the servers are connected to the stable memory of their datacenter. The broker is the customer representative in the cloud environment. It allocates resources for applications/services on multiple VMs in order to fulfill requests of the client, it negotiates also with the resource provider (the cloud computing) to obtain the most cost-effective resources. The contract between the broker and the resource provider is represented by the service level agreement (*SLA*). The client, the broker and the cloud datacenters are connected by a network.

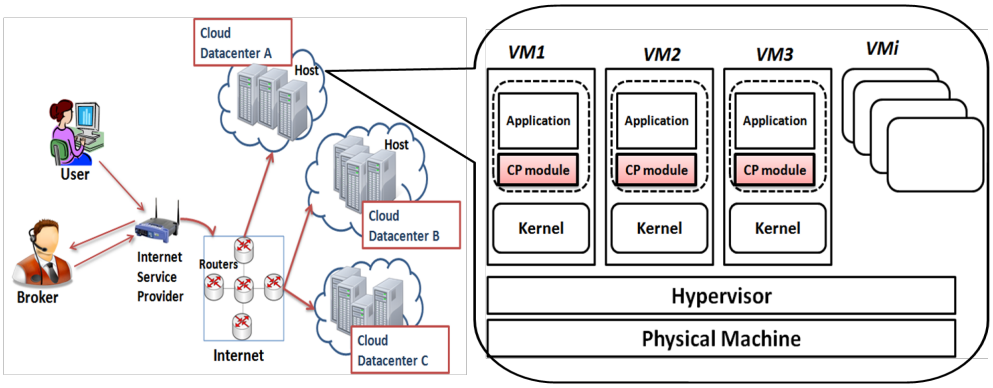

FIG. 3.1. *System model*

The application models in cloud computing can vary from multi-tier web applications such as e-commerce to scientific applications (parallel and data-driven applications and workflows) [15]. Typically such applications consist of several tasks, which communicate with each other. A task consists of some computation and communication phases. Communications among tasks create dependencies [15], [19]. The workflow is used to indicate the temporal relationship among tasks and to present their dependencies. In order to show how the workflow is presented in the system, we used an oriented graph as shown in Figure 3.2.

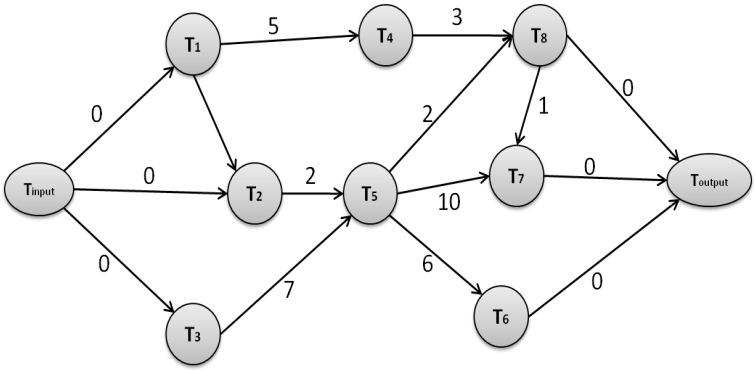

FIG. 3.2. *Workflow example with eight tasks*

The oriented graph consists of eight tasks from T_1 to T_8, and two dummy tasks: t_{input} and t_{output}. The number above each arc (directed edges) shows the estimated dependency factor between the corresponding tasks. In other words, if task T_1 sends five messages to task T_4 then the $ArcWeight = 5$. The average communication rate μ is calculated by the formula 3.1.

$$\mu = \frac{\sum_{l=0}^{r} ArcWeight_l}{r} \tag{3.1}$$

where r is the number of arcs in the workflow. The number of tasks executed in parallel depends on the number of available VMs.

Since the tasks are dependent to each other by their communications, transit and orphan messages can exist during the creation of checkpoints which makes the independent checkpointing inadequate to ensure a consistent state. However, coordinated checkpointing is very expensive in term of overhead and energy consumption. To resolve this problem, we used an adaptive time based coordinated checkpointing *ATCCp*.

The work in [6] is similar to our proposition but it assumes that VMs clocks are perfectly synchronized, which may not be true in real system. The work in [6] ensures also only a simple consistency without taking into consideration the transit messages. Our *ATCCp* deals with both transit and orphan messages to ensure a strong consistency with the minimum overhead and without any need for clock synchronization.

4. Time based coordinated checkpointing. Time based coordinated checkpointing is a combination between coordinated and uncoordinated checkpointing, it is proposed to avoid the problems of each checkpointing strategy. To facilitate the understanding of our work, we present in Table 4.1 the principal symbols cited and used in the rest of this paper.

TABLE 4.1
Time based checkpointing parameters

Symbol	Signification
ρ	Timer drift
T	Initial timer value
MD	Maximum deviation
D	Initial deviation of checkpointing interval
ED_C	Effective deviation for the consistency
ED_R	Effective deviation for the recoverability
csn	Checkpoint sequence number
t_{max}	Maximum delivery time of a message
t_{min}	Minimum delivery time of a message
FS_i	Vector of sending flags of VM_i
T_C	Checkpointing interval

Time based coordinated checkpointing is based on the idea that if the VMs create their checkpoints at the same time (without any deviation), orphan messages will not exist [17], [18], [23]. However, the clock synchronization among all the VMs is impossible in real systems. To overcome this problem, time based coordinated checkpointing uses a timer instead of real clock and it assumes that the VMs have loosely synchronized clocks. It assumes also that all the VMs timers are approximately synchronized with a deviation from real time of some value ρ, which presents the clock drift.

If a timer is started on each of two VMs at the same time, and each VM has clock drift rate equal to ρ, the timers will expire at $2\rho T$ seconds apart, where T was the initial timer value and $10^{-8} < \rho < 10^{-5}$. Assuming this, the clocks will exhibit a maximum drift of $2\rho nT$ after N checkpoint intervals [19]. Time based coordinated checkpointing ensures a strong consistency if it satisfies two conditions: consistency condition and recoverability condition.

4.1. Consistency condition. The consistency condition ensures that the system will not create any orphan message during the checkpointing process. So the checkpointing must ensure that if the "sent event" of a message m is not recorded in the checkpoint file of the source, the "receive event" will not be recorded also in the checkpoint file of the destination [12]. To deal with orphan messages and satisfy the consistency condition, many solutions are proposed and implemented in the literature.

In [6], [17], [25], the authors assume that the timers (real or logical) are perfectly synchronized. In this case, all the VMs will create their checkpoints at the same time. In Case 1 of Figure 4.1, VM_0 and VM_1 create their checkpoint $C_{0,0}$ and $C_{1,0}$ at the same time (in $C_{i,j}$: the i is the VM identifier and j is the checkpoint csn). If VM_0 sends a message m to VM_1 after the checkpointing, it is sure that the message will be received after the checkpoint of VM_1, so m will never be an orphan message. However, the perfect timer synchronization among VMs is not possible in real systems.

To avoid the synchronization condition, the papers [18], [19], [24] take into account the timer deviation ρ. In this case, orphan message can be created. The second case of Figure 4.1 illustrates a possible situation where VM_0 creates its checkpoint $C_{0,1}$ at time t_1 and then sends a message *m1* to VM_1. After receiving *m1*, VM_1 creates the checkpoint $C_{1,1}$. So *m1* will be an orphan message and the consistency is not assured. If VM_0 did not send *m1* during the timer deviation (until VM_1 creates its $C_{1,1}$), the consistency will be assured.

Based on this observation, the papers [18], [19], [24] propose to block the message sent during a certain period after the checkpointing. Using time based checkpointing assumptions, it is possible to specify approximately

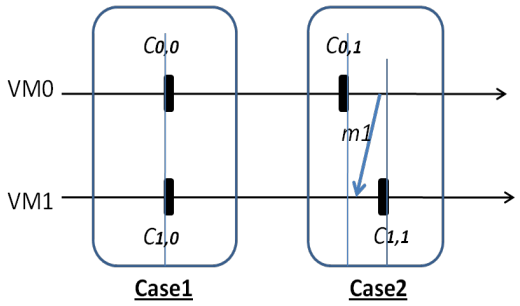

FIG. 4.1. *Possible cases of consistency condition*

the time interval after the checkpointing where potential orphan messages can be created. If $\rho \neq 0$ then the Maximum Deviation can be calculated by Formula 4.1:

$$MD = D + 2\rho nT \tag{4.1}$$

where D is the initial deviation of checkpointing intervals. A message sent during MD after the checkpointing can cause an inconsistency. The MD period can be minimized by taking into account the transmission time of the message, so the effective deviation of consistency ED_c will be (See Formula 4.2):

$$ED_c = MD - t_{min} \tag{4.2}$$

where t_{min} is the minimum delivery time of a message inside the server. If $MD = 0$ then the timers are fully synchronized (case 1 of Figure 4.1). Preventing a VM to send messages during ED_c period (by buffering this message until the end of ED_c) can resolve the problem. However, blocking the communication increases the response time and the SLA violation.

Both the timer synchronization and the blocked communication decrease the system performances. To ensure the consistency, another solution proposes the use of piggybacked messages [20], [21], [22], [23]. In this case, each message will piggyback some extra data and the receiver will decide to create or not the checkpoint according to this data. One of the piggybacked data is the checkpoint sequence number (*csn*) of the local VM. The *csn* is an integer number that will be incremented each time the VM creates its checkpoint file. In case of time based coordinated checkpoints, the *csn* in all VMs of the server must be the same since all these VMs create their checkpoints using the same checkpointing interval T_C.

So, if a VM receives a message where the piggybacked *csn* of the message is greater than the local *csn*, the receiver VM knows that the sender has already created its checkpoint before sending the message. In this case and to avoid the creation of orphan message, the VM receiver must create its checkpoint before dealing with the received message (it buffers the message, creates the checkpoint and then computes this message).

In case 2 of Figure 4.1, the message *m1* will piggyback the $csn = 1$, when VM_1 receives this message, it compares the local $csn = 0$ with the *csn* of the message, since the local *csn* is less than the sent *csn*, VM_1 will know that VM_0 has created its checkpoint before sending *m1*, so VM_1 will be forced to create the checkpoint $C_{1,1}$ before dealing with *m1*. This strategy increases the delivery time of messages.

4.2. Recoverability condition. The recoverability condition ensures that the system will not create any transit message during the checkpointing process. The checkpointing must ensure that if the "receive event" of a message m is recorded in the checkpoint file of the destination, the "send event" will also be recorded in the checkpoint file of the source [12].

Using an illustrating example in Figure 4.2, transit messages can exist even if the VMs create their checkpoints at the same time. In case 1 of Figure 4.2, both VM_0 and VM_1 created their checkpoints at the same time, VM_0 has sent *m0* before the checkpointing but VM_1 has received *m0* after its checkpoint which makes *m0* transit.

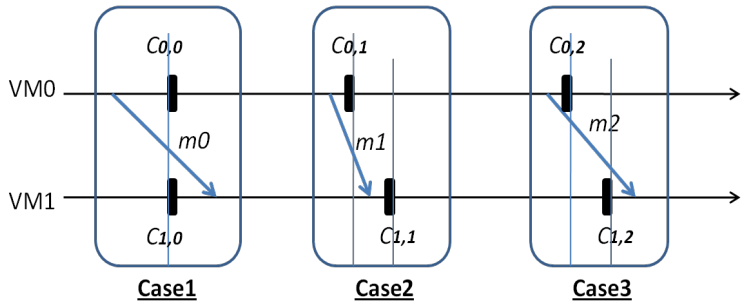

FIG. 4.2. *Possible cases of recoverability condition*

The clock deviation also does not always cause a transit message. In case 2 of Figure 4.2, the message *m1* is not transit since its send and receive events are not recorded in the global state. So the existence of transit message is related more to the transfer time of the message (time from sending a message to receiving it) compared to the checkpointing interval. The case 3 of Figure 4.2 presents the same situation of case2. However in case 3, the transfer delay of *m2* was greater than the transfer delay of *m1* in case 2, so *m2* is received after the checkpointing of VM_1 but sent before the checkpointing of VM_0 which makes it transit message.

To deal with the problem of transit messages, the paper [24] proposes to block the communication a certain period just before the checkpointing time. The transit message is a message sent before the checkpointing of a VM_i and received after the checkpointing of VM_j. So if we prevent VM_i to send any message that can be received after the checkpointing of VM_j, the problem will be resolved.

Using time based checkpointing assumptions and adopting the same reasoning of blocked communication used to resolve the problem of case2 in Figure 4.1, we can estimate the time interval that can create potential transit messages. This interval named effective deviation of recoverability ED_R is calculated by Formula 4.3:

$$ED_R = MD + t_{max} \tag{4.3}$$

where t_{max} is the maximum delivery time of a message in the server. However, blocking the communication always adds an extra overhead to the system.

To avoid blocking communications, the papers [18], [20], [23] use the send-messages logging. In this strategy, the VM logs all the messages sent. In case of rollback of the receiver, it can ask the sender to re-sent these messages. But how many messages must be logged and until when. The only period that can create potential transit message is ED_R before the checkpointing time [20], [24], [25]. So logging the message sent during this period can resolve the transit message problem. We summarized all the used techniques that ensure the consistency and the recoverability conditions in Figure 4.3.

5. The contribution. In the previous section, we cited and explained the existing time-based coordinated checkpointing works. The majority of these works deal with the consistency by blocking the communications or piggybacking the messages. There are also some works that do not consider important parameters such as: timer drift, checkpointing overhead, message transfer time. Some works ensure only a simple consistency. So each strategy suffers from some imperfections and does not deal with some problems. And here comes our contribution as a solution of these imperfections and problems.

5.1. Adaptive Time based Coordinated Checkpointing (*ATCCp*). The proposed contribution in this paper is an adaptive time based non-blocking coordinated checkpointing (*ATCCp*). *ATCCp* ensures a strong consistency without overheating the system and without synchronizing the timers or blocking the communications. *ATCCp* has three phases: Initiator selection, Checkpointing and Resynchronization.

5.1.1. Initiator selection Phase. In this phase, each server (host) selects an initiator of checkpointing. The initiator is the most powerful and less busy VM in the server. In *ATCCp*, the role of the initiator is the general resynchronization and the checkpointing control.

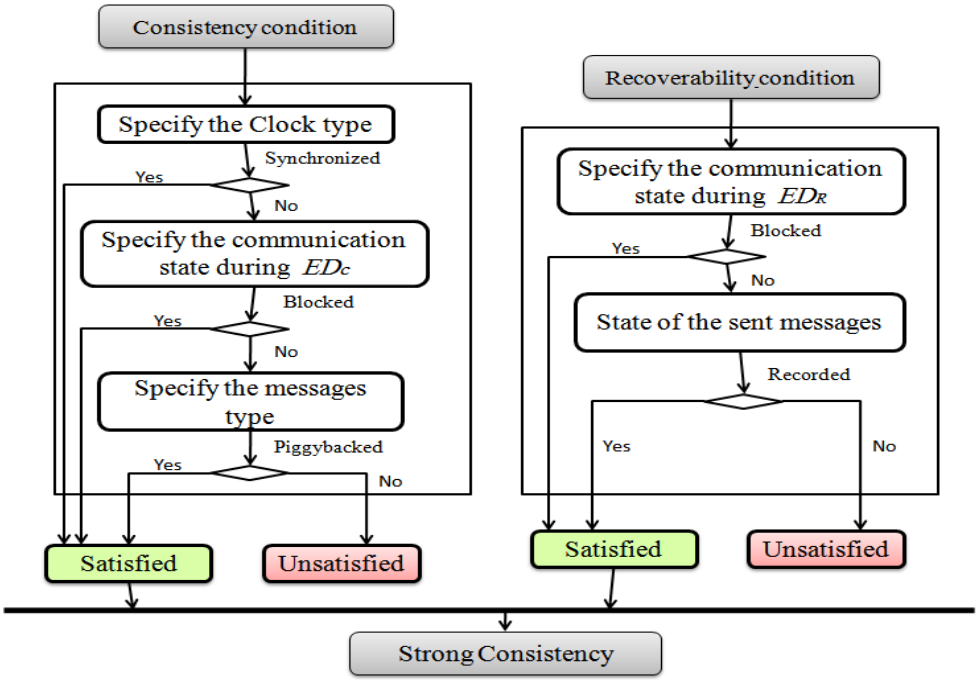

FIG. 4.3. *Time based checkpointing strategies*

5.1.2. Checkpointing Phase. Periodically, each VM creates its checkpoint independently without any control message exchange. *ATCCp* ensures a strong consistency by dealing with both consistency and recoverability conditions. Since *ATCCp* is not blocking or synchronizing checkpointing, recording and piggybacking messages are the only solutions to ensure a strong consistency. Contrary to the previous cited works, our approach ensures the desired consistency with the minimum overhead by reducing the number of recorded and piggybacked messages.

In *ATCCp*, the messages sent during ED_C after the checkpointing are potential orphan messages. To avoid blocking the communication during ED_C (such as the strategies in [18], [19], [24]), when a VM sends a message during ED_C to another VM (in this case the "send event" is not recorded), the VM receiver buffers the message and creates its checkpoint file, then it computes the message. In this case, the "receive event" will be also unrecorded in the checkpoint file of the receiver. However, not all the messages sent during ED_C will force the receivers to create their checkpoints. The VM destination is forced to create the checkpoint if the *csn* of the message (actually it is the *csn* of the source VM) is greater than the local *csn* of VM destination.

Besides the *csn*, the piggybacked message will also contain the "time to next checkpoint TNC" representing the rest of the time before the next checkpoint of the source. The goal of this information is the resynchronization. To reduce the number of piggybacked message in *ATCCp*, a VM_i piggybacks only its first application message sent during ED_C (after it has create its checkpoint) to a VM_j.

Each VM_i uses a Boolean vector FS_i of N size, where N is the number of VMs in the server. The FS_i is used to identify the first message sent to a VM destination, when VM_i sends a message to VM_j then $FS_i[j] = 1$. After each checkpointing the vector is initialized ($FS_i[j] = 0$ / $i \neq j$). The second computation message sent to the same destination can not force the checkpointing creation because the source and the destination have the same *csn*.

In *ATCCp*, the potential transit messages in VM_i are those sent during the period ED_R. So only the messages sent during this period will be stored in the checkpoint file and VM_i continues its execution without blocking its communications. In case of rollback, the transit message will be detected and the source will send it again to the destination. However in paper [20], the author has proved that if we use the same checkpointing interval (even with deviations) and if $T_c + MD > t_{max}$ (the maximum difference between the sender *csn* and

receiver *csn* is 1) then it is sufficient to store only the messages sent during the last checkpointing interval.

5.1.3. Resynchronization Phase. In time based checkpointing approachs, the clock deviation increases with $2\rho nT$ after n checkpoint intervals [18], [19]. The goal of the resynchronization phase is to reinitialize the value of ρ. If $\rho = 0$ then the Maximum deviation will be: $MD = D + 2\rho nT = D$ where D is the initial deviation of checkpointing intervals. The synchronization can be executed by the initiator or the ordinary VM.

- The initiator can start the resynchronization when the deviation MD exceeds a certain threshold or the size of logged messages exceeds the buffer size of VMs. In this case, the initiator sends special messages to VMs to synchronize their timers.
- An ordinary VM can use its piggybacked message to minimize the deviation between the local timer and the destination timer. So when the VM receives the piggybacked message, it uses the *csn* to decide to create or not the checkpoints and it uses the synchronization data (*time to next checkpoint TNC* and the message transfer delay) to minimize the clock deviation.

The combination between the initiator and the VM resynchronization decreases considerably the overhead. The initiator synchronization overheads the server by control messages but it decreases the resource consumption, i.e. the number of possible orphan and transit messages will be reduced since the ED_R and ED_C are reduced. The VM synchronization adds more data on the piggybacked messages but it decreases the initiator resynchronization rate.

However, if the timer of the sender is faulty, the erroneous timer information will be spread to the receiver. Besides, since the transmission delay between the sender and the receiver is variable, the timer information from the sender may not reflect the correct situation when the message finally arrives at the receiver. To achieve more accurate timer synchronization, we can utilize the timers in the initiator as an absolute reference. The Figure 5.1 illustrates a part of *ATCCp* process (only the consistency condition and the resynchronization of an ordinary VM) in two communicating VMs (VM_i and VM_j). The green arrow presents a communication message sent from VM_i to VM_j.

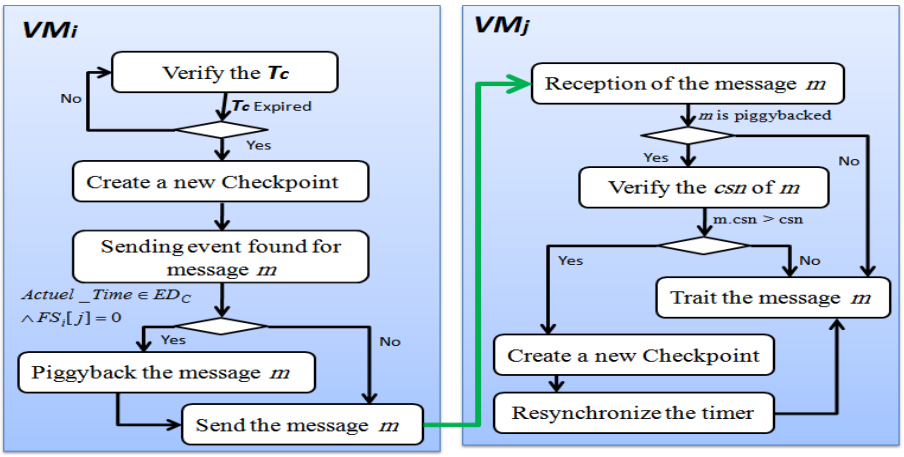

FIG. 5.1. *ATCCp process in two VMs*

Besides the checkpointing technique, it is proven that 70% of checkpointing overhead is caused by the storage phase (called the checkpointing latency), i.e. the time needed by the VM to store its checkpoint file in the stable memory. During the storage phase, the VM suspends its running task to execute the storage which means that the task will be delayed and therefore the whole workflow will be delayed also.

To minimize the storage time, we used soft checkpointing. When the VM creates its checkpoint file, it sends it to the initiator (the file) and continues immediately the running task. When the initiator receives all the checkpoints files, it sends these files to the stable memory. The initiator can use I/O strategies such as data sieving and collective I/O [26] to improve the storage time.

Since *ATCCp* is a server checkpointing technique, we used an enhanced version of *VIOLIN* topology [2]. The *VIOLIN* (virtual networked environment) consists of multiple VMs connected by a virtual network. VMs of the same server are connected by *VIOLIN* switches running in each server. In the classical *VIOLIN* proposed in [2], the checkpoints are taken by *VIOLIN* switches from outside VMs using the strategy proposed in [3]. Although communications pass through the *VIOLIN* switch, each VM is responsible to manage its sent/received messages. *VIOLIN* topology is destined for the fault tolerance, but it is very sensitive to the *VIOLIN* switch failure. If a virtual switch in *VIOLIN* fails (in case of in-transient failures), the checkpointing will be impossible and the cloud will lose its reliability.

To address this problem in *VIOLIN* topology we combined between the host topology of Figure 3.1 and the classical *VIOLIN* by implementing a checkpointing module in each VM. One VM will be selected as *VIOLIN* switch and in case of failure; another VM can replace it immediately. *ATCCp* does not need any extra-control messages or extra-resource, so its implementation will not need major modifications.

To facilitate the checkpointing process, we suppose that the *VIOLIN* switch is always the checkpointing initiator because it has a global view about the existing VMs in the host (server). In case of any need of control (for example: specifying the maximum deviation time or starting the resynchronization), the *VIOLIN* switch can forward the control messages without losing time in collecting data about the actual situation. Besides that, the *VIOLIN* switch uses the server clock to solve the timer problem.

5.2. Analysis and comparative study. To analyze our contribution, we used the example of Figure 5.2.

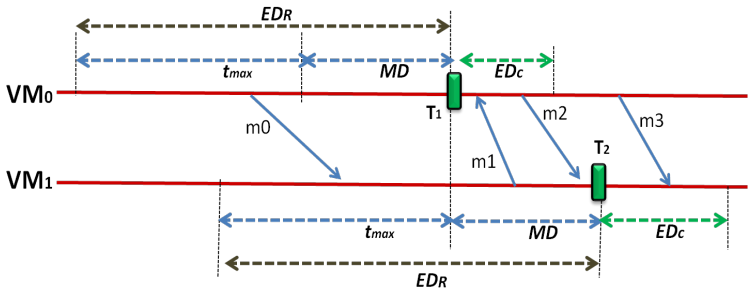

FIG. 5.2. *ATCCp parameters with an example*

Figure 5.2 illustrates all the time intervals used in time based coordinated checkpointing (see Table 4.1). It presents also an example of the checkpointing process in two VMs .

- Each VM estimates its ED_R and ED_C.
- VM_0 and VM_1 create their checkpoint files (*T1* and *T2* respectively) independently
- Each VM stores the messages sent during its ED_R in the checkpointing file. Those messages are probably transits. In case of rollback, the transit message will be sent again to its destination. In Figure 5.2, the message *m1* is transit and it will be stored in the checkpoint file of VM_1. In case of failure and rollback, the VM_0 requests VM_1 to resend *m1*, and *m1* is already stored in the checkpointing file of VM_1.
- The messages sent during ED_C of the source can be an orphan message, so if the message is sent for the first time to a destination, that message will be piggybacked by some data (specially the *csn*) and it will force the destination to create its checkpoint before the checkpointing (only if *csn* of the source is greater than the *csn* of the destination). In this case, the message will be regular and the *csn* of the source and the destination will be the same. In Figure 5.2, the VM_0 sent *m2* to the VM_1 during its ED_C and since this message is the first message sent to VM_1 after the last checkpoint T1, it will be piggybacked by some data. At the reception of $m2(Source_{csn} = 2 > Destination_{csn} = 1)$, the VM_1 creates its checkpoint T2 before treating this message (not illustrated in the Figure 5.2) and it will also resynchronize its timer. If VM_0 sends a second message to VM_1 during ED_C, the message will not be piggybacked with extra-data and it will not force the destination to create its checkpoint file since *csn* of VM_0 and *csn* of VM_1 are equal.

Our approach solves many problems of time based coordinated checkpoints proposed in literature and it does not omit the majority of parameters. Table 5.1 presents a comparison between our *ATCCp* and the other time based coordinated checkpointing cited in this paper.

TABLE 5.1
Comparison between time based checkpointing

Paper	ρ	**Transfer time**	**Timer prob**	**Blocking**	**Sys type**	**CP type**	**Synch**	**Recov**	**Piggy-back**
[17]	No	No	No	Yes	Dist	H	SM	No	No
[18]	Yes	Yes	No	Yes	Dist	H	SM	Yes	No
[19]	Yes	Yes	No	Yes	Dist	H	SM	Yes	No
[20]	Yes	No	No	No	Mob	S/H	SM	Yes	Yes
[21]	Yes	Yes	Yes	No	Mob	S/H	CM	Yes	Yes
[22]	Yes	No	No	No	Mob	H	CM	No	Yes
[23]	Yes	No	No	No	Mob	S/H	CM	Yes	Yes
[24]	Yes	Yes	No	Yes	Mob	H	CPM	Yes	No
[25]	No	Yes	No	No	Dist	H	SM	Yes	Yes
[6]	No	No	No	No	Cloud	H	CM	No	Yes
ATCCp	Yes	Yes	Yes	No	Cloud	S/H	CM/ SM	No	Not always

The comparison criteria are: considering the timer drift (ρ), considering the transfer time of messages (*Transfer time*), dealing with timer problems (*Timer prob*), if the system freezes communications or not (*Blocking*), the system type where the checkpointing strategy was implemented for (*Sys type*: Mob (Mobile), Dist(Distributed) or Cloud), the resynchronization tools (*Synch* represents the type of message where the synchronization is piggybacked: SM (Special messages), CM (Computing messages) or CPM (Checkpointing messages)), ensuring the recoverability condition (*Recov*), using the piggybacked messages (*Piggyback*). And finally the type of checkpoint files: Hard (*H*) or Soft (*S*). In hard checkpointing, the VM stores its state in the stable memory directly. In the soft type, the VM stores its checkpoint in the local memory or sends it to a storage manager and continues its execution immediately. The S/H means that the checkpointing uses both soft and hard checkpoint files.

6. Performance evaluation. In this section, we evaluate the performances of our checkpointing scheme using *CloudSim* simulator [11]. The *CloudSim* is currently the most sophisticated discrete event simulator for Clouds; it is an open source, scalable and low simulation overhead simulator [11]. It is used in many papers to simulate the scientific workflows [31], [32], [33].

The experimentations in [32] prove that *CloudSim* has the best accuracy in workflows performances. The accuracy is defined as the ratio between the predicted performances using a specified metric and the real performances.

We did not compare our *ATCCp* with the other time based coordinated checkpoints because the *ATCCp* is a solution of classical time based checkpointing problems, which makes it clearly better than those strategies. However, we studied the *ATCCp* behavior compared to coordinated checkpointing *CCp* used in [1], [16], [35] and independent checkpointing *ICp* used in [15], [28], [36]. The goal is proofing that our approach ensures a strong consistency with the minimum cost (like the *CCp*) and with the minimum overhead (like the *ICp*). Simulation parameters are presented in Table 6.1.

6.1. Makespan vs Number of VMs. In the first experiment, we measured the impact of number of checkpoints on the makespan (scalability) for the three checkpointing strategies *ATCCp*, *ICp* and *CCp*. We used the parameters cited in Table 6.1, except that the cloudlet length is fixed at 1000 MIPS, μ=50, the number of VMs is 40 and the checkpointing interval: $100 < T_C < 600s$. The makespan is measured by Formula 6.1.

$$makespan = max_{T_i \in T} ExectTime(T_i) \tag{6.1}$$

where T_i is the task i in the running workflow T. And $ExecTime$ is the execution time of that task (Task i).

TABLE 6.1
Simulation parameters

Parameter	Value
Number of VM per server	10-100
Server BW	1 Gega bit per second
Cloudlet number (Tasks)	1500
Cloudlet length	100-12000 MIPS
communication rate μ	2-100
Checkpoint interval$CP_{Interval}$	100-500 second
Failure rate λ	2 to 5 per period

The results are presented in Figure 6.1. Incrementing the checkpointing number increases the makespan in all the strategies (*ATCCp*, *CCp*, *ICp*). *ATCCp* uses soft checkpointing and it does not need any coordination except in case of resynchronization, which makes its performances similar to the *ICp* with an average difference of 10%.

In our work, we present the $SLA_{Violation}$ by the overhead caused during the checkpointing [8]. The *SLA* is a contract between customers and service providers of the level of service to be provided [1]. So the customer can precisely specify the margin for acceptable overhead (20% for example). If the overhead caused by the checkpoint exceeds the margin, then a SLA violation will be detected. In this experiment, $SLA_{Violation}$(ATCCp)=13%, $SLA_{Violation}$ (CCp)=27% and $SLA_{Violation}$ (ICp)=11.78%.

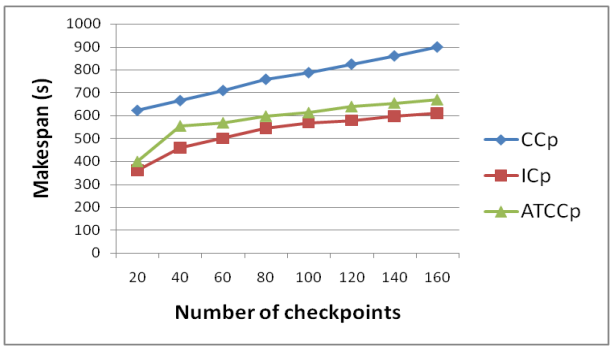

FIG. 6.1. *Makespan vs Number of checkpoints*

6.2. Cost vs Number of checkpoints. As the cost is a very important parameter in cloud environment, we used the parameters of the first experiment to analyze the cost of all checkpointing strategies. The total cost is calculated using Formula 4.3:

$$TotalCost = \sum_{i=1}^{k} Cost(T_i) + \sum_{j=1}^{m} Cost(CP_j) \tag{6.2}$$

where k is the number of tasks in the workflow and m is the number of checkpoints created during the workflow execution. $Cost(T_i)$ is the cost of running the task T_i in the specified resource, it includes the communication cost (running the send/receive events). The cost of the j^{th} round of checkpointing is $Cost(CP_j)$ and it includes the coordination cost, the state recording cost and the storage cost.

According to the results illustrated in Figure 6.2, the cost of *CCp* is less than the cost of *CCp* and *ICp* (specially in case of high checkpointing rate). The only checkpointing communication needed in *ATCCp* is during the resynchronization so the coordination cost will be reduced. Also the use of VMs resynchronization minimizes the timer drift, so the periods ED_C and ED_R will also be reduced. In this case, the piggybacked messages and the number of transit messages stored in checkpoint files will be reduced, which means less storage

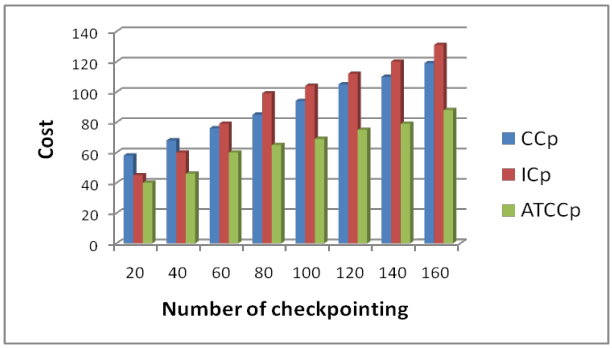

FIG. 6.2. *Cost vs Number of checkpoints*

cost and less bandwidth consumption. Also in *ATCCp*, only one checkpointing state is needed to ensure a correct rollback.

In *CCp*, the server must record all the sent messages to avoid the transit messages (ensuring the recoverability condition) and the initiator must coordinate with the VMs in each checkpointing round, which increases the cost. The cost of *ICp* was significantly increased when the checkpointing rate increased because this type of checkpointing needs to store all the checkpoint files to ensure the correct rollback (domino effect). So the *ICp* cost is due to the checkpointing latency (storage).

6.3. Communication rate vs cost. Communication rate μ represents the dependency degree between tasks of the workflow. This dependency causes a major problem in checkpointing techniques. In this experiment, we measured the cost of the *ATCCp* and *CCp* checkpointing in the system with different μ. The *ICp* is not affected by the communication rate since no coordination is needed.

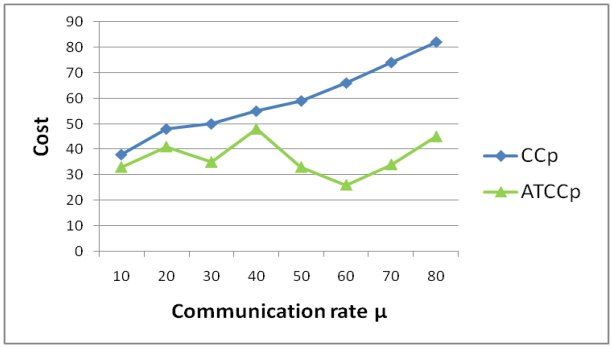

FIG. 6.3. *Communication rate Vs cost*

The results (See Figure 6.3) prove that increasing the communication rate can improve the *ATCCp* performances since the communication among VMs (specially during the ED_C) decreases the timer drift (using the piggybacked messages). So the period ED_R will be reduced and the initiator resynchronization frequency will be decreased. In *CCp*, a high μ implies more storage space needed to ensure the recoverability condition. The number of piggybacked messages in *CCp* will increase since this checkpointing does not block the communications. In our *ATCCp*, piggybacked messages are sent only during ED_C (See Section 5).

VIOLIN topology has improved both checkpointing techniques with almost 14.3% since the initiator is *VIOLIN* switch (it has a global view about the communication among VMs), so no time is needed to select an initiator, or to collect data about communications or VMs states in the server.

6.4. Rollback duration Vs failure rate. In this experiment, we fixed all the parameters cited in Table 6.1, but the failure rate λ is varied from 3 to 24 per period. Our goal is to measure the time needed for the

rollback in case of failure. The performances of *CCp* and *ATCCp* are approximately the same because both strategies need only the last stored checkpoint file to resume the work. However, *ICp* has to ensure the correct state among all the stored checkpointing files, which can lead to the domino effect and therefore the rollback duration will increase (See Figure 6.4).

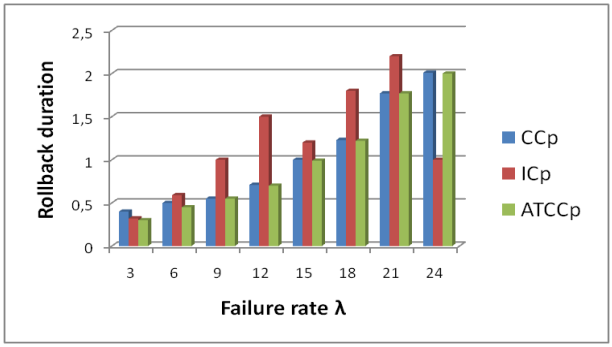

FIG. 6.4. *Rollback duration Vs failure rate*

7. Conclusion. When computationally-intensive workflows are executed, fault handling is very important, since the failure of a single component might lead to an abandonment of the entire workflow. This may lead to the loss of the stability and the reliability in the system. Since both the customer and the cloud computing are bound by the SLA contract, the choice of the checkpointing technique is very critical.

In this paper we proposed a fault tolerance service to tolerate failures using checkpointing technique. Our approach is a time-based coordinated checkpointing *ATCCp*. It ensures a strong consistency with the minimum control messages and without blocking the VMs communications. *ATCCp* uses the *VIOLIN* topology and the soft checkpoint to improve the checkpointing performances.

To evaluate our approach, we compared it with the most popular checkpointing techniques: coordinated and uncoordinated checkpointing. The experimental results prove that *ATCCp* omits the problems of classical approaches. It decreases the overhead, minimizes the SLA violation and ensures a rapid rollback. Our work focuses on intra-server checkpointing. However, *ATCCp* can be easily extended to support the inter-server checkpointing by using multi-level checkpointing (Communication induced checkpointing at servers level and *ATCCp* at VMs level). In the future work, we will use a real platform such as eucalyptus [30] to implement the *ATCCp*.

REFERENCES

[1] M. Zhang, H. Jin, X. Shi and S. Wu, *VirtCFT: A Transparent VM–Level Fault-Tolerant System for Virtual Clusters*, IEEE Proceeding of the 16th International Conference on Parallel and Distributed Systems (ICPADS), 8–10 Dec. 2010, Shanghai, pp. 147–154.

[2] A. Kangarlou, P. Eugster and D. Xu, *VNsnap: Taking Snapshots of Virtual Networked Infrastructures in the Cloud*, IEEE Transactions on Services Computing, 5(4): 484–496, 2012.

[3] F. Mattern , *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation*, Journal of Parallel and Distributed Computing, 18(4): 423–434, 1993.

[4] S. Di ,Y. Robert, F. Vivien, D. Kondo, C–L. Wang and F. Cappello, *Optimization of cloud task processing with checkpoint–restart mechanism*, International Conference for High Performance Computing, Networking, Storage and Applications (SC2013), Nov. 17–22, 2013, Dever, Colorado, USA.

[5] D. Nguyen and N. Thoai, *EBC: Application–level migration on multi–site cloud*, The International Conference on Systems and Informatics (ICSAI), 19–20 May, 2012, pp. 876–889.

[6] H. Hui, Z. Zhan, W. Bai–Ling, Z. De–Cheng and Y. Xiao–Zong, *A Two–level Application Transparent Checkpointing Scheme in Cloud Computing Environment*, International Journal of Database Theory and Application, 6(2): 61–71, 2013.

[7] O. Ben–Yehuda, A. Schuster, A. Sharov, M. Silberstein and A. Iosup, *ExPERT: Pareto–Effcient Task Replication on Grids and a Cloud*, IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS), 21–25 May, 2013, pp. 167–178.

[8] R. Jhawar, V. Piuri and M. Santambrogio, *Fault Tolerance Management in Cloud computing : A System–Level Perspective*, IEEE Systems Journal, 7(2): 288–297, 2013.

[9] M.N.O. Sadiku, S.M. Musa and O.D. Momoh, *Cloud computing: Opportunities and challenges*, Potentials IEEE Journal, 33(1): 34–35, 2014.

[10] W. Zhao, Y. Peng, F. Xie and Z. Dai, *Modeling and Simulation of Cloud Computing: A Review*, IEEE Asia Pacific Cloud Computing Congress (APCloudCC), 14–17 Nov. 2012, Shenzhen, pp. 20–24.

[11] R. Calheiros, R. Ranjan, A. Beloglazov1, C. De Rose and R. Buyya, *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software: Practice and Experience journal, 41(1): 23–50, 2011.

[12] G. Cao and M. Singhal, *On coordinated checkpointing in distributed systems*, IEEE Transactions on Parallel and Distributed Systems, 9(2): 1213–1225, 1998.

[13] I. Egwutuoha, S. Chen, D.Levy, B. Selic and R. Calvo, *A Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the Cloud*, Second International Conference on Cloud and Green Computing (CGC2012), 1–3 Nov. 2012, Xiangtan, pp. 268–273.

[14] L. Arockiam and G. Francis, *FTM–A Middle Layer Architecture for Fault Tolerance in Cloud Computing*, IJCA Special Issue on Issues and Challenges in Networking, Intelligence and Computing Technologies, ICNIT 2012, No.2, pp. 12–16.

[15] T. Nguyen and J-A. Desideri, *Resilience Issues for Application Workflows on Clouds*, In Proceeding of the 8th International Conference on Networking and Services (ICNS2012), Mach 2012, Sint–Maarten (NL).

[16] P. Palaniammal and R. Santhosh, *Failure Prediction for Scalable Checkpoints in Scientific Workflows Using Replication and Resubmission task in Cloud Computing*, International Journal of Science, Engineering and Technology Research (IJSETR), 2(4): 985–991, 2013.

[17] S. Neogy, A. Sinha and P. K. Das, *Checkpointing with synchronized clocks in distributed systems*, International Journal of UbiComp, 1(2): 64–91, 2010.

[18] N. Neves and W.K. Fuchs, *Coordinated checkpointing without direct coordination*, IEEE International Computer Performance and Dependability Symposium (IPDS98), 7–9 Sept. 1998, Durham, NC, pp. 23–31

[19] P.K. Suri and M. Satiza, *System Progress Estimation in Time based Coordinated Checkpointing Protocols*, International Journal of Computer Applications, 52(11): 1–6, 2012.

[20] N. Neves and W.K. Fuchs, *Adaptive Recovery for Mobile Environments*, Communications of the ACM, 40(1): 68–74, 1997.

[21] C–Y. Lin, S–C. Wang and S–Y. Kuo, *An Efficient Time–Based Checkpointing Protocol for Mobile Computing Systems over Mobile IP*, Mobile Networks and Applications, 8(6): 687–697, 2003.

[22] K. Singh, *On mobile checkpointing using index and time together*, World Academy Science, Engineering and Technology, Vol. 26, 2007, pp. 144–151.

[23] J. Surender, S. Arvind, K. Anil and S. Yashwant, *Low Overhead Time Coordinated Checkpointing Algorithm for Mobile Distributed Systems*, Computer Networks & Communications (NetCom) in Lecture Notes in Electrical Engineering (LNEE), Vol. 131, 2013, pp. 173–182.

[24] M. Tripathy and C.R. Tripathy, *A new co–ordinated checkpointing and rollback recovery scheme for distributed shared memory clusters*, International Journal of Distributed and Parallel Systems (IJDPS), 2(1): 49–58, 2011.

[25] B. Gupta and S. Rahimi, *A Novel Low-Overhead Recovery Approach for Distributed Systems*, Journal of Computer Systems, Networks and Communications, Vol. 2009, 2009, pp. 1–8.

[26] J. Fu, M. Min, R. Latham and C. D. Carothers, *Parallel I/O Performance for Application–Level Checkpointing on the Blue Gene/P System*, Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS), in conjunction with IEEE International Conference on Cluster Computing (Cluster), 26–30 Sept. 2011, Austin, TX, pp. 465–473.

[27] I.I. Yusuf and H. W.Schmidt , *Parameterised Architectural Patterns for Providing Cloud Service Fault Tolerance with Accurate Costings*. In Proceedings of the 16th International ACM SIGSOFT symposium on Component–based software engineering (CBSE13), June 17–21, 2013, Vancouver, BC, Canada, pp. 121–130.

[28] L. Ramakrishnan and D. A. Reed, *Performability Modeling for Scheduling and Fault Tolerance Strategies for Scientific Workflows*. In Proceedings of the 17th international symposium on High performance distributed computing (HPC08), 23–27 June 2008, Boston, MA, USA, pp. 23–34.

[29] M. Wang, L. Zhu and J. Chen, *Risk–aware Checkpoint Selection in Cloud-based Scientific Workflow*, Second International Conference on Cloud and Green Computing (CGC2012), Xiangtan, Hunan, China, November 1–3, 2012, pp. 137–144.

[30] E. Caron, F. Desprez, D. Loureiro, and A. Muresan, *Cloud computing resource management through a grid middleware: A case study with DIET and eucalyptus*, IEEE International Conference on Cloud computing (CLOUD2009), 21–25 Sept. 2009, Bangalore, India, pp. 151–154.

[31] M. Sudha and M. Monica, *Investigation on Efficient Management of workflows in cloud computing Environment*, International Journal on Computer Science and Engineering (IJCSE), 2(5): 1841–1845, 2010.

[32] W. Chen and E. Deelman, *WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments*. In Proceedings of the IEEE 8th International Conference on E–Science (E–SCIENCE '12), October 8–12, 2012, Chicago, IL, USA, pp. 1–8.

[33] R. N. Calheiros and R Buyya, *Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication*, IEEE Transactions on Parallel and Distributed Systems (TPDS), No. 99, 2013, pp. 1–10.

[34] A. Gupta, L. Kale, F. Gioachin, V. March, C. Suen, B. Lee, P. Faraboschi, R. Kaufmann and D. Milojicic, *The Who, What, Why and How of High Performance Computing Applications in the Cloud*, Technical Reports, HP Laboratories, HPL–2013–49, 2013. pp. 1841–1845.

[35] K.L. Keville, R. Garg, D.J. Yates, K. Arya and G. Cooperman, *Towards Fault–Tolerant Energy Efficient High Performance Computing in the Cloud*, IEEE International Conference on Cluster Computing (Cluster12), 24–28 Sept. 2012, Beijing, pp. 622–626.

[36] B. Nicolae and F. Cappello, *BlobCR: Virtual Disk Based Checkpoint–Restart for HPC Applications on IaaS Clouds*,

Journal of Parallel and Distributed Computing , 73(5): 698–711, 2013.
[37] P. Zaspel and M. Griebel, *Massively parallel fluid simulations on Amazons HPC cloud*, First International Symposium on Network Cloud Computing and Applications (NCCA2011), 21–23 Nov. 2011, Toulouse, France, pp. 73–78.
[38] E. Walker, *Benchmarking Amazon EC2 for high–performance scientific computing*, LOGIN , 33(5): 18–23, 2008.
[39] L. Ramakrishnan, P.T. Zbiegel, S. Campbell, R. Bradshaw, R.S. Canon, S. Coghlan, I.Sakrejda, N. Desai, T. Declerck, and A. Liu, *Magellan: experiences from a science cloud*, In Proceedings of the 2nd international workshop on Scientific cloud computing (ScienceCloud11), June 8, 2011, San Jose, California, USA, pp. 49–58.
[40] B. Meroufel, G. Belalem, *Collaborative Services for Fault Tolerance in Hierarchical Data Grid*, International Journal of Distributed Systems and Technologies (IJDST), 5(1): 1–21, 2014.

Edited by: Florin Fortiş
Received: Mar 1, 2014
Accepted: Jun 24, 2014

DOI 10.12694/scpe.v15i2.982
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 2, pp. 169–185. http://www.scpe.org

REGULATED CONDITION-EVENT MATRICES FOR CLOUD ENVIRONMENTS

RICHARD M. WALLACE[1], PATRICK MARTIN[2], AND JOSÉ LUIS VÁZQUEZ-POLETTI[3]

Abstract. Distributed event-based systems (DEBS) are networks of computing devices. These systems have been successfully implemented by commercial vendors. Cloud applications depend on message passing and inter-connectivity methods exchanging data and performing inter-process communication. Both DEBS and Clouds need time-coordinated methods of control not dependent on a single time domain. While DEBS have specific implementation languages for complex events, Cloud systems do not. Clouds and DEBS have not yet presented an explicit separation of temporally based event processing from computations. Using a regulated, isomorphic, temporal architecture (RITA), a specific language and separation of temporal event processing from processing computation is achieved. RITA provides a functional programming style for developers using familiar language constructs for integration with existing processing code without forcing the developer to work in multiple coding paradigms requiring extensive "glue code" allowing coding paradigms to work together. This paper introduces RITA as a guarded condition-event system that has explicit separation of event processing and computation with constructs allowing integration of time-aware events for multiple time domains found in Cloud or existing distributed computing systems.

Key words: Cloud; Distributed networks; Distributed applications; Data Abstraction; Complex Event Processing; Condition-event Matrix, DEBS, Event Model

AMS subject classifications. 68M14, 68M20, 68N19

1. Introduction. RITA is an event processing architecture initially developed as a proprietary product in the telecommunications industry between 1998 and 2000 [29] allowing federated and distributed systems to communicate without undue interruption of intensive computing applications. An open source implementation for RITA is being developed by the authors of this paper. This paper precedes the release of this code base to introduce RITA concepts and facilitate understanding of RITA concepts.

The roots of RITA came from real-time avionics control software [30, 16] and with the current move to have cloud systems handle time critical processing, so RITA is now relevant to cloud systems. These cloud systems have reached a processor interconnect scale requiring carefully controlled intra- and inter-processor communication just as avionics and telecommunications systems require [2]. RITA focuses on the real-time processing and optimal communication needs of intensive computing applications by controlling time-aware and data value sensitive guards to prevent unnecessary interruption thus allowing more CPU time for intensive computing applications. We will show that RITA is a novel service for high performance computing applications that have migrated to cloud environments.

RITA is designed for event based, quantitative, time critical communication for computationally intensive applications which include communication networks, modeling and simulation, financial forecasting and trading systems, computational chemistry, and drug discovery as obvious applications. RITA can also be applied to workflow systems that are instance-intensive. These systems are unlike computationally intensive systems as they have thousand of instances per second of concurrent, often relatively simple, workflow instances.

RITA is a novel minimal set of canonical event transform types that are formally defined. These transform types are used for specification of *á priori* time and data values relevant to event-based data transformations. RITA is a compact event system where its elements are used in combination to build larger event processing systems. Thus, by establishing a condition-event matrix control for a DEBS, it is possible to have data controls written in a functional programming form and express the imperative form as a separate control structure. The C language is used in RITA due to its high degree of familiarity by developers.

During RITA development JMS [5] was just being written and CORBA [18], IBM MQ [12], TIBCO Rendezvous [27], BEA Tuxedo [21] (now owned by Oracle Corp.), and other message oriented middleware products were in their initial stages of use. Over time these and other DEBS have matured and have become general purpose messaging systems with the majority of their messaging development focused on publish and subscribe

[1]Universidad Complutense de Madrid, Spain, wallacerim@aol.com
[2]Queen's University, Kingston, Ontario, Canada, martin@cs.queensu.ca
[3]Universidad Complutense de Madrid, Spain, jlvazquez@fdi.ucm.es. The authors would like to thank the ServiceCloud project (MINECO TIN2012-31518).

semantics. This has been well understood and codified by the Object Management Group over the ensuing years [19]. RITA does not use CORBA semantics, is agnostic to inter-process communication (IPC) and data transport methods, and does not prescribe a specific interface thus allowing it to be compatible with communication products and techniques such as activeMQ, Websphere MQ, shared memory, sockets, OpenMPI, and others.

The structure of the paper starts with an informal description of the event model types in §2 followed by the formal semantics described in §3 which is a foundation for the condition-event matrix described in §4, the explicit time components in §5, an example RITA structure in §6, relevancy to cloud computing in §7, related work in §8, and concluding with future work in §9. The paper defines event originators, the event model, and event processing as per the reference model discussion in [22].

2. Event Model. The RITA event model is based on a set of canonical state transformations that can be combined to create complex event interactions. The three canonical types are "spike," "set-at," and "transitional" which are informally defined in the following subsections. Time is shown as t and event state is shown as σ.

Each type occurs over a Δt for the event space time-frame in which it occurs. Time is infinitely divisible, countable, and continuously and monotonically increasing, thus any incremental units of measurement are only sample points of infinitely countable time. This definition provides for any discussion of an event "lifetime." Any perceived event lifetime is defined by the latency of its occurrence and its subsequent observance. This latency is graphically shown in Figures 2.1, 2.2, and 2.3 showing the canonical types state change using electrical square-wave graphics. The slope shown in these figures is a named state change ($\sigma_n \rightarrow \sigma_{n\pm1}$) of an event and this illustrates that a state change has an interval of time in which its state is not known while time continues to increment, albeit by an infinitesimal amount, during a state change. Event sequencing requires that there be a sufficient lifetime so that a sequence of events can create a queue. If an event has an insufficient lifetime, or if the next event arrival overwrites the current event, then no queuing can occur. Lifetimes are detailed for each event type in the following subsections.

The remainder of this section builds the logical foundation of event semantics by describing the temporal mechanics of the canonical event types. This is then further detailed in §3 where each type is formally described, cumulatively producing a basis for the RITA notation shown in §6 which is enforced in the runtime environment by following these formally defined event types and temporal specifications. This formalism is the core of the RITA concept. We believe RITA to be an advancement to the current state of the art for distributed, federated cloud applications through use of the condition-event matrix detailed in §4.

2.1. Spike Event. The spike event graphic in Figure 2.1 occurs at a Δt asymptotic to zero. Figure 2.1 has prior time shown as time in the past until event execution, $-\infty \cdots t_n$, where t_n is the "now" time of the event occurring that causes state change from σ_1 to σ_2 for a subdivision of time t_n shown as $\Delta t_{n_{0..1}}$. At the end of this asymptotic to zero time, the state changes back to the original σ_1 state for time $t_{n+1} \cdots \infty$. Events of this type are considered periodic "heartbeat" events. This event type can be used for counted threshold limits, keep-alive notification, or request for service. This event type can not be queued and has no lifetime.

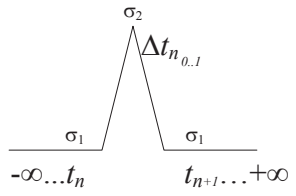

FIG. 2.1. *"Spike" Event. Vertical axis is state (axis omitted)*

2.2. Set-At. The "set-at" event graphic in Figure 2.2 is a permanent state change to a new state. State σ_n remains unchanged until an event occurs and a permanent state change to state $\sigma_{n\pm1}$ occurs at the next time increment t_{n+1}. Past and future time are as described in §2.1. This event type can be used for one-time

events such as system initialization and system termination. As shown in Figure 2.2, this is not the transitional event type as there is no ability for a transition from state σ_n to return after transitioning to state $\sigma_{n\pm1}$. This event type has an infinite lifetime and can be queued.

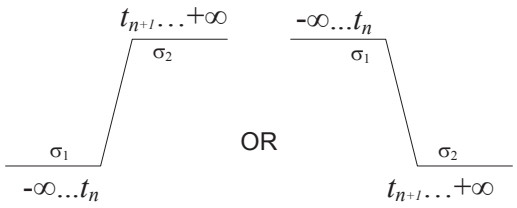

FIG. 2.2. *"Set-At" Event. Vertical axis is state (axis omitted)*

It is important to discuss the difference between a *set-at* event and a *transitional* event. While the wave forms used to illustrate the canonical events may appear the same as shown in Figures 2.2 and 2.3, note the definite differences in the time element t where, as is explained in Subsection 2.2, a *set-at* event can transition only once during the lifetime of the system; whereas a *transitional* event can oscillate between states σ_1 and σ_2 at any frequency greater than one. This temporal difference is very important in constructing a system of events without ambiguity as to the semantics of the intent of a state change.

2.3. Transitional. The transitional event graphic in Figure 2.3 occurs over some period of measurable time with an event-sensitive entity perceiving a state change which separates this event type from the "spike" event type. Past and future time is as described in §2.1. This event type is semantically dense as each increment of time can result in a different state. The semantic meaning is compounded based on the frequency of state change and for how many sampling intervals of t the state remains constant. This event type has a limited lifetime, albeit the transition between $\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_1$ can be as long as the system needs it to be, thus is can appear as a set-at event type, but it is not because it can revert while set-at can not revert to its σ_1 state. Queuing of this event type requires prioritized time-based queuing that can be dynamically edited as higher priority events occur or as queued events expire or both.

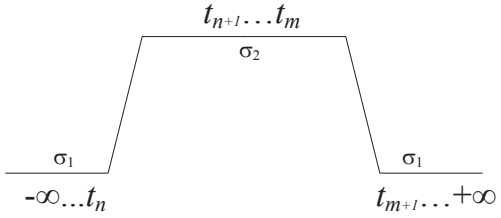

FIG. 2.3. *Transitional Event. Vertical axis is state (axis omitted)*

2.4. Event Type Use. Using these event types, RITA processes events as shown in Figure 2.4. Given one or more events E, conditions C, guards G, and actions A, the system has $1 \cdots n$ inputs to a precondition-event matrix evaluating events against specific guards. Guards are a proper subset of conditions ($G \in C$). If the guard evaluates to *True*, the action step comprised of $1 \cdots n$ conditions on that event, $C_1(E) \ldots C_k(E)$, are evaluated. The results of the action step are $0 \cdots n$ outputs, based on the evaluation of condition vectors, to a post-condition event matrix. If the post-condition event guard matrix evaluates to *True*, the output action data – now seen as input event data – is input to another precondition-event guard matrix. In actual use, the definition of guards for precondition and post-condition matrices are compressed in the event evaluation chains so that the trace,

$$\begin{aligned} &G_{pre}(E) \rightarrow A \rightarrow G_{post}(E) \rightarrow \\ &G_{pre}(E) \rightarrow A \rightarrow G_{post}(E) \cdots \end{aligned} \tag{2.1}$$

has the post-condition guard being the precondition for the next action step:

$$G_{pre} \rightarrow A \rightarrow G_{post_{pre}} \rightarrow A \rightarrow G_{post_{pre}} \cdots \tag{2.2}$$

As shown in transition chain 2.2 each event system can be chained to other event systems.

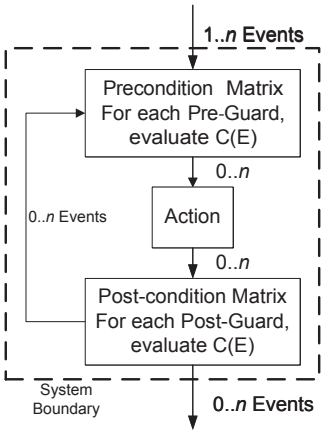

FIG. 2.4. *Event Processing*

3. Formal Event Model. Building on the informal description given in §2, a definition for the formal event specifications and formal condition-event matrix is given. For each of the three event forms (i.e. types) the formal definition of the *precondition* and *post-condition* are formally defined. The action step, as shown in transition chain 2.2 above, is not formally defined. The precondition allows an event to initiate an action step (calculation of application data) in the system. A post-condition dampens output from the action step by suppressing output that does not evaluate to $True$ in the post-condition. Only post-conditions create new events, or propagation of existing events, in the system. Guards and conditions evaluate specified application data as being time, summation, or delta critical.

Time critical information is data which must reach the computational process at fixed times usually expressed as some delta time. Summation critical information is data that is summed to form a value that triggers an event. Delta critical information is data that must be significantly different from its previous value in order to trigger an event.

3.1. Spike Event. The spike event form has the precondition specification of:

$$\exists E : \forall C_{pre}(E) \rightarrow True \tag{3.1}$$

There exists an event such that a precondition on the event will evaluate to *True*. The spike event form has the post-condition specification of:

$$\begin{aligned} &\exists E : \forall C_{pre}(E) \rightarrow True \therefore \\ &\exists A(E) \Rightarrow \exists E' \ni C_{post}(E') \rightarrow True \end{aligned} \tag{3.2}$$

Proposition 3.2 asserts the precondition and then states therefore there exits an action for the event implying that there exists some output event E' such that the post-condition C_{post} evaluates to $True$.

3.2. Set-At Event. The set-at event form has the precondition specifications based on an initial condition value:

$$\begin{aligned} &\exists E : \forall C(E)_{(-\infty \ldots t_{n-1})} \equiv False; \\ &\exists A_{t_n}(E) : C_{pre}(E) \rightarrow True \langle \forall \Delta t \rangle \end{aligned} \tag{3.3}$$

There exists an event where, under all conditions, E results in $False$. At some time t_n there exists an action, A, where the precondition for the event results in $True$ over the sequence of all increments of time. Temporally, using basic propositional linear temporal logic [15], this is:

$$\Box[\Box P(E) \Rightarrow \Box P(E')] \tag{3.4}$$

This states that there is always an implication of $P(E)$ to $P(E')$ for all predicates of E and E'.

The set-at event is a terminal state transition. This means that once the set-at event form has happened, it is an individuate action and the post-condition is *NULL* as an external source to the entire system is required to change the state once the event occurs to transform the set-at event state.

3.3. Transitional Event. The transitional event form has the precondition specification of:

$$\exists E : C_{pre}(E) \rightarrow True\langle \Delta t_1 \cdots \Delta t_n \rangle \in \forall \Delta t) \tag{3.5}$$

There exists an event where the precondition results in $True$ for a subset of all time. The post-condition is specified as:

$$\begin{aligned} &\exists C_{pre}(E) \rightarrow True \therefore \{\exists A(E) \mid True\langle \forall \Delta t_{t_1 \ldots t_n} \rangle\} \\ &\Rightarrow \exists E' \ni C_{post}(E') \rightarrow True \end{aligned} \tag{3.6}$$

There exists a precondition on an event evaluating to $True$; therefore there exists an action for the event such that it is $True$ comprehended over a sequence of time implying that there exists an output event for the post-condition that evaluates to $True$.

4. Condition-Event Matrix. The RITA condition-event matrix is comprised of a system of three matrices that are evaluated by row; expressed as:

$$\begin{bmatrix} G_1 \\ \vdots \\ G_n \end{bmatrix} \bullet \begin{bmatrix} C_1(E_1) \cdots C_k(E_1) \\ \vdots \\ C_n(E_n) \cdots C_{n,k}(E_n) \end{bmatrix} \bullet \begin{bmatrix} op_{1,1} \cdots op_{1,k-1} \\ \vdots \\ op_{n,1} \cdots op_{n,k-1} \end{bmatrix} \rightarrow \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} \tag{4.1}$$

Given trace 2.2, the chaining of systems of condition-event matrices requires that guards be part of preconditions only. As shown in Equation 4.1, a vector, V, is defined as V: $(C_n(E_n) \cdots C_k(E_n))$, a "row." The condition vector is comprehended, post guard evaluation at a single time t, across the condition resultants with the operators $\wedge$ "`and`," $\vee$ "`or`," $\oplus$ "`xor`," and $\neg$ "`not`" is expressed as:

$$C_1(E_1)\ op_1\ C_2(E_1)\ op_2\ C_3(E_1) \cdots op_{k-1}\ C_k(E_1) \tag{4.2}$$

The resultant vector, R, if $True$ indicates that the output event E' be propagated to the vector of guards in the next condition-event matrix. The separation of logical control occurs using the G_n, C_n, and op_n components. Internal to C_n components is where data control dispatch calls occur to application code. If a condition position is empty in the matrix the postfix op_n is considered *NULL*. Thus a row may may be sparsely populated and the empty C_n and *NULL* $op_{n,k}$ allows for sparse condition vectors as these matrix elements do not have to be evaluated.

5. Explicit Time. All guard and condition expressions for RITA have time expressed in delta or summation time. Currently these expressions use a restricted subset of C language syntax due to its familiarity with developers. Allowed constructs are *IF-THEN-ELSE*, *SWITCH* where alternatives must be inclusive of all values passed by the Guard condition, relational operators, no pointers or locally declared storage is allowed, and only boolean stack values are allowed as return values with the required form: <*condition_name*>*_TRUE* or <*condition_name*>*_FALSE*. As the <*condition _name*> is known to the system, the post-fix *TRUE* or *FALSE* is a name decoration that RITA enforces.

Vector variable data that represents time values uses three time functions ensuring that the condition-event matrix does not have to rely on user manipulated time functions. RITA uses the *evaluate_time*() function in the

Guard or Condition code. Its function signature is: *int evaluate_time(<vector variable>,[BEFORE | AFTER | NOW], <delta>)*. The additional functions *create_time*(), and *get_time*() are only used in application code external to the event engine. The event engine system time is uniform throughout evaluation of the guard and its condition vector for a specific RITA system. Time domains can, and do, vary across instances of RITA systems. RITA handles these variant time domains by design.

Real-time implementations need fine-grain time evaluation. For the event engine *evaluate_time*() function, comparison of vector variables of time type use the highest precision time for the computer architecture executing it. Special timing access is needed beyond the default programming API to the system time function. Since 2005 high precision time such as the High Precision Event Timer (HPET) hardware timer used in Intel chip sets [14] and, since 2013, nanosecond time resolution with the HPET library is available [7]. Within a cloud environment, systems may also rely on precision hardware timers like those available from Symetricom (http://www.symmetricom.com).

The resultant of the *evaluate_time()* function is *TRUE* or *FALSE*. The special enumeration values of **BEFORE**, **AFTER**, and **NOW** are used to determine the comparison of the system time with the vector variable time. The **NOW** enumeration must have a decimal delta of zero. The delta value is a numerical constant that is a positive floating point value with the precision needed and supported by the system hardware timer.

The *get_time()* function is used to examine the time value set in the *create_time()* function. Assignments to vector variables are allowed. Vector variables can be used in multiple event matrices. Chaining of event matrices ensures that the event value is modified in a deterministic method, without race conditions, and that the tuple of $\{event, value\}$ is propagated through-out the system of condition-event matrices resulting in a final event consumption. If a user causes race conditions by having multiple matrices modify a vector variable in parallel, the pre-processor will alert the user. These restrictions ensure that each condition-event matrix executes in the same sequence given the same event sequences and event values.

6. Putting the Parts Together. The preceding sections have described the RITA event system and the mechanics of the system. A brief code fragment showing the structure of the RITA system description notation is presented in order to link the event formalism in §2 and §3 to code which is the specification notation created by the pre-processor as shown in Figure 6.1 where the RITA specification is parsed by the Guard and Condition pre-processor into a compiled language. The application, RITA translated code, and specific libraries are then compiled and linked into one image. While this may appear to be a a monolithic implementation, it is not as each RITA system can be matched to a application program function and thus, at the most granular level, one has independent communicating processes *á la* Communicating Sequential Processes by Hoare [11].

In Listing 3 the elements of a RITA *system, guard, vector,* and *resultant* are seen. These elements are described for each line in Listing 3:

- Lines 1-2: Linkage to application code functions that can run as detached processes.
- Lines 6-9: Declaration of events and variables for *TEST_SYSTEM_1*. Note the two events that have a named canonical event form, and their data type as an event occurrence has data associated with it.
- Lines 11-25: Declaration of conditions. Note that *INITIAL_CHECK* contains an application call to *user_function_1* that runs as a detached process while *user_function_2* is a blocking call for *FLOW*. The latter, while allowed, is not a preferred method of invoking application code.
- Lines 27-37: Declaration of guards. As $G \in C$, these look very much like conditions as they should. The primary usage is to short-circuit the evaluation of conditions in a vector. As such these are very simple gating controls.
- Lines 40-52: Declaration of vectors. Note the order of guard, condition(s), and result as these are the implementation of the condition event matrix. The boolean operation attached to each condition matches the op_n in the condition event matrix. There may be no result from an evaluation. See the null condition for *VECTOR_2* at line 49. Note the output from *VECTOR_1* is *SystemOn* and *null* in *VECTOR_2*. These two events stimulate further action in the RITA system.
- Lines 59-104: This is the second system in the RITA event system configuration. Vector *System2_Looper* has the event *Update* propagating a new value of the event. The constructs in *TEST_SYSTEM_2* are like *TEST_SYSTEM_1* and their descriptions are not repeated.
- Lines 106-116: This is the RITA systems linkage. Note these are parallel statements as each system is a

detached process. The symbol "`<-`" is read as "Assign output from the right-hand system, or event, to the left hand system." Note that multiple right-hand systems may be specified as input to a left-hand system using this notation: "`system1 <- system2, system3`" as an example. This notation is similar to concurrent signal assignment statements in VHDL [13].

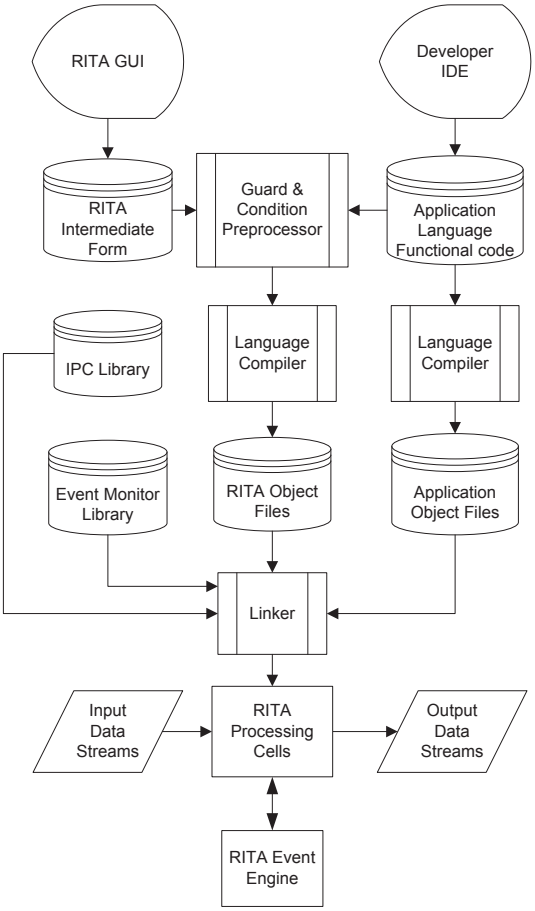

FIG. 6.1. *RITA Infrastructure.*

LISTING 3
RITA Code Fragment

```
1 with "../application/user_app.hpp";
2 use  "user_function_1", "user_function_2";
3
4 system TEST_SYSTEM_1
5 {
6   event(trans, bool) : SystemValue1(false);
7   int4         : Requests(0);
8   time         : FlowDuration("500ms");
9   float8       : FlowRate(0.0);
10
11  condition INITIAL_CHECK {
12    if( SystemOn == true ) {
13      user_function_1();
```

```
14        return INITIAL_CHECK_TRUE;
15      }
16      return INITIAL_CHECK_FALSE;
17    } // INITIAL_CHECK
18
19    condition FLOW {
20      FlowRate = user_function_2(FlowDuration);
21      if( FlowRate < 5 ){
22        return FLOW_TRUE; }
23      else {
24        return FLOW_FALSE; }
25    } // FLOW
26
27    guard SystemOn {
28      if( !SystemOn ) { break VECTOR_1; }
29      return SystemOn_TRUE;
30    } //SystemOn
31
32     guard Update {
33      if( Update <= 10 ) {
34        return Update_TRUE;
35      }
36      return Update_FALSE;
37    }
38
39    // SystemOn is true at start. Vector will activate.
40    vector VECTOR_1 {
41      guard           : SystemOn;
42      condition(and) : INITIAL_CHECK;
43      condition(or)  : FLOW;
44      result          : SystemOn1(true);
45    } // VECTOR_1
46
47    vector VECTOR_2 {
48      guard           : Update;
49      condition(and) : null;
50      condition(or)  : INITIAL_CHECK;
51      result          : CheckEvents(true);
52    } //VECTOR_2
53
54  } // TEST_SYSTEM_1
55
56  // ++
57  // TEST_SYSTEM_2
58  // --
59  system TEST_SYSTEM_2
60  {
61    int2 : update_value(0);
62    int2 : tmp_update(0);
63    int2 : Requests(0);
64
65    condition SetUpCheckEvents {
66      if( update_value <= 10 ){
67        tmp_update = update_value + 6;
68      }
69      tmp_update = 0;
```

```
70      return SetUpCheckEvents_TRUE;
71    }
72
73    condition AddUpdate {
74      update_value = update_value + 1;
75      return AddUpdate_TRUE;
76    }
77
78    guard SystemOn1 {
79      if( SystemOn1 == true) {
80        return SystemOn_TRUE; }
81      else {
82        break START_CheckEvents;}
83    }
84
85    guard CheckEvents {
86      if( Requests < 10) {
87        Requests = Requests + 1;
88        return CheckEvents_TRUE;
89      }
90      return CheckEvents_FALSE;
91    } //CheckEvents
92
93    vector START_CheckEvents {
94      guard           : SystemOn1;
95      condition(and) : SetUpCheckEvents;
96      result          : Update(tmp_update);
97    }
98
99    vector System2_Looper {
100     guard           : CheckEvents;
101     condition(and) : AddUpdate;
102     result          : Update( update_value + 1);
103   }
104 } // TEST_SYSTEM_2
105
106 control {
107   event(setat, bool) : SystemOn(false);
108   event(setat, bool) : SystemOn1(false);
109   event(spike, int4) : CheckEvents(0);
110   event(trans, int4) : Update(0);
111
112   begin
113     TEST_SYSTEM_1 <- SystemOn(true);
114     TEST_SYSTEM_2 <- TEST_SYSTEM_1;
115     TEST_SYSTEM_1 <- TEST_SYSTEM_2;
116   end
117 }
```

The listing demonstrates the usage of each canonical event form, the condition event matrix elements (*Guards, Vectors, Operators,* and *Resultants*) from Equation 4.1. RITA specifications are converted into executable code via the development procedure shown in Figure 6.1. This code is compiled with the application and becomes the event control mechanism benefiting overall throughput of the system.

In Listing 3 the *control* block at line 112 starts all concurrent/parallel event execution. At lines 40, 47, 93, and 99, concurrent and parallel execution of each *vector* begins. The *vector* structure executes in a sequential manner with initial evaluation of the *guard*, then one or more *conditions* with a final *result*. In the case of

conditions and *results* a **null** statement can be used. Using null in a *condition* clause acts as a placeholder and the condition becomes a non-operation for vector evaluation. Using null in a *result* clause indicates that the *vector* has no output event which is useful when a vector needs to terminate event propagation or a vector is only activated to start concurrent/in parallel calls to user functions.

7. Cloud Computing. Cloud systems can be separated into four major layers:
IaaS: Infrastructure as a Service — physical/virtual machines and hypervisor
NaaS: Network as a Service — network/transport services between systems
SaaS: Software as a Service — access to application software and databases
PaaS: Platform as a Service — IaaS, base O/S, run time services, and NaaS
RITA is a run time service in the PaaS layer. The advantages of a RITA in the PaaS layer are:

- Preventing application unbounded priority inversion by reducing the communication interrupts that lead to this problem.
- As an event propagator it is designed for partitioned, communicating processes in a "share all," "shared partial," or a "shared nothing" environment supporting parallelism without heavy use of synchronization primitives for publication/subscription (pub/sub) systems or message passing interfaces.
- High performance worker threads for "scatter gather" configurations can use the RITA bifurcation of communication and computation to improve performance with a high performance PaaS IPC.
- Canonical event forms define the data communication needed for a distributed application making all communication explicit and drastically reducing communication traffic loads and errors.

Cloud computing has processes executing autonomously and independently communicating with each other. This communication can quickly increase to a level that decreases system throughput. As mentioned in §1, detailed in §8, communication currently in use depends on uniform, monolithic communication mechanisms following a pub/sub methodology for tight-cluster, grid, or single systems. These systems are dependent on a single homogeneous time domain that requires an inordinate amount of effort handling latency issues in a monolithic time domain for federated, distributed cloud systems. This latency can cause side-effects due to imperative code managing events. Events are either missed and must be resent, if possible, or the imperative code induces more latency by direct queue management from the application code.

Declarative coding in RITA allows side-effect free conditions that are separate from the imperative application code. Using RITA it is possible create a series of processes that can be (a) event sensitive by using only the canonical event forms (i.e., spike, set-at, and transitional), (b) temporally sensitive, based on local time domain and triggered by canonical events, and (c) value sensitive through delta value comparisons. This results in each RITA cell being semi-autonomous so unexpected behavior can either be ignored or can trigger a recovery in RITA processing cells as required. RITA is designed to work in a distributed and autonomous environment. A discussion of cloud computing and its dependencies on networks [1] observes that the issues of network traffic are becoming limiting factors in cloud computing systems. RITA can reduce unnecessary IPC traffic and so improve application performance as described in [30] and [29]. Other DEBS do not have the clean demarcation between the declarative and imperative portions of an application and the ability to naturally enforce functional programming in a non-functional specific language which is why Lisp, Scheme, and Haskell have never really surpassed C, C++, Java, and other imperative languages in commercial popularity. RITA processing stacks shown in Figure 7.1 give an example of how separate physical systems can have multiple RITA systems (i.e. a processing stack) mapped and how each system communicates with each other via an IPC interface and event engine that enforces event form semantics.

7.1. RITA Innovation and Performance. RITA allows multi-CPU (single board computers or multi-core) distributed systems used in cloud computing to have increased application mode time and a decreased time in IPC mode improving throughput. As stated in §1 these classes of processing are improved in a cloud environment by not having processing interrupted to check if a significant data value has arrived, a specific time period has passed, or having to poll or message other applications to know if a significant event has occurred thus reducing context switch "thrashing." Measurements of using RITA in a distributed telecommunications environment are given in Figures 7.2 to 7.9. In Figure 7.1, each *RITA System* in the RITA specification has its own application space and executes concurrently or in parallel with its peers — depending on the hardware architecture — on a "physical" system that may be a multi-core CPU, virtual machine, single-board computer,

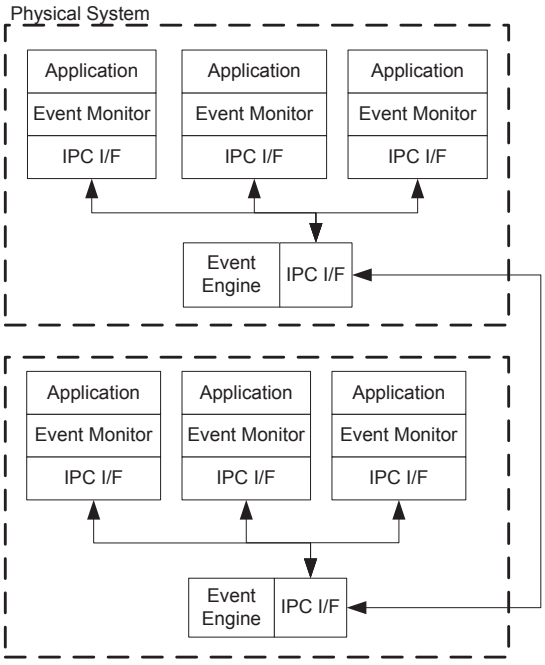

FIG. 7.1. *Multiple RITA Systems*

or separate computing machine. Common distributed systems operate in this fashion but RITA is the only one that prevents unnecessary IPC from reaching computationally or instance-intensive applications.

Figures 7.2 to 7.9 present data collected for an application controlled IPC and a RITA controlled IPC for the Mass Markets Billing and SHERIFF (Statistical Heuristic Engine to Reliably and Intelligently Fight Fraud) systems which was a joint venture between MCI and British Telecom [23]. This heterogeneous grid implementation was comprised of multiple IBM/VM LPARs, an multi-node VAXCluster, and multiple IBM/AIX HACMP RS/6000s. The figures show the beneficial result of using RITA. RITA reduces the decision logic processing time needed by application code.

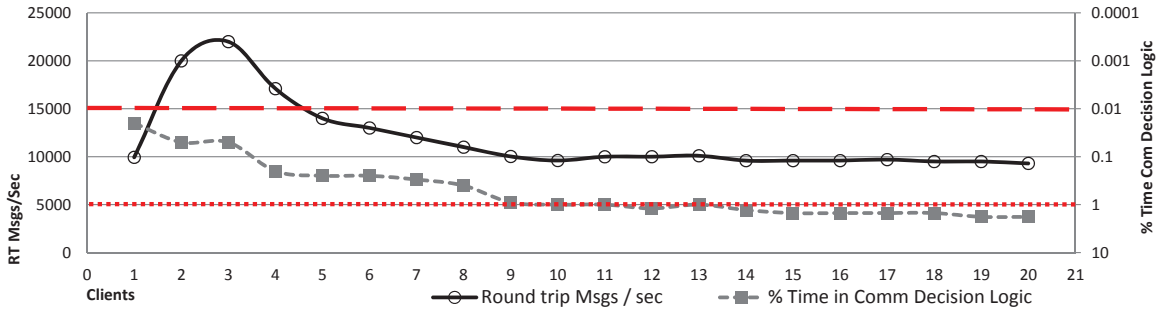

FIG. 7.2. *Local client, Non-RITA, Non-Persistent.*

Operational requirements set by the MCI and BT corporations for these systems were 100% data integrity and 100% data delivery for all message payloads with ACID[1] guarantees for both in memory or persistent message data forms. For transient data, "eventually consistent" BASE[2] semantics were required for system

[1]Atomicity, Consistency, Isolation, Durability
[2]Basically Available, Soft state, Eventual consistency

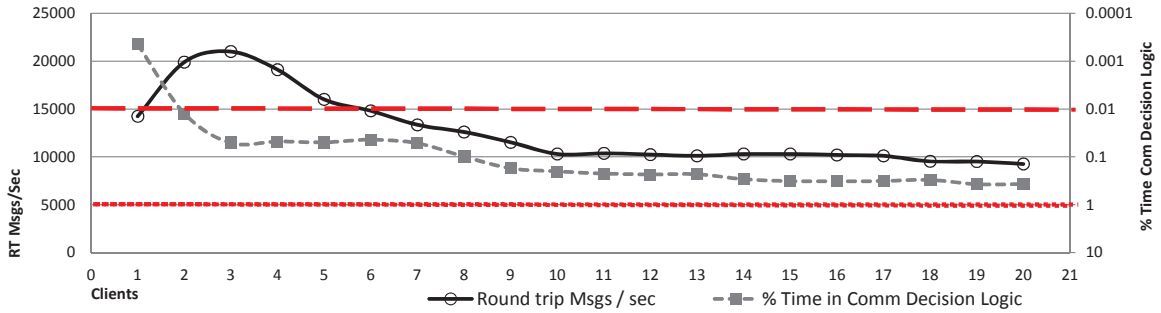

FIG. 7.3. *Local client, RITA enabled, Non-Persistent.*

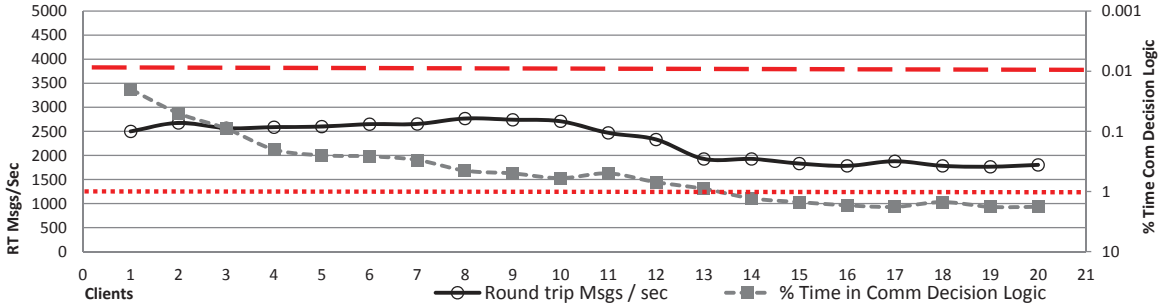

FIG. 7.4. *Local client, Non-RITA, Persistent.*

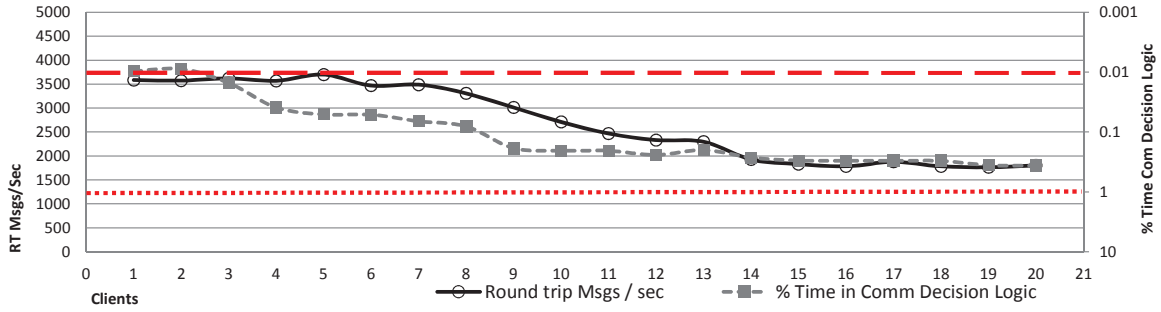

FIG. 7.5. *Local client, RITA enabled, Persistent.*

informational messages, logging, or diagnostic messages. Tests used MQSeries 5.2 local client in-memory queues and server channel queues. Local client and server channel experiments were run with and without RITA, and with and without persistence. Test were run 1,000 times and averaged.

Results show an 80% time reduction in communication decision logic processing thus applications had less probability of context switch thrashing. The increase in the number of messages per second produced by RITA shows that the IPC subsystem had less application interrupt. RITA did have some effect in improving round-trip message volumes but, while significant, it was considered a beneficial side-effect as round-trip performance is determined by network congestion and IPC buffering and queuing and not directly controlled by RITA.

The graphs are paired for non-RITA and RITA use. Figure 7.2 is paired with Figure 7.3 and so on, showing round-trip volumes in messages per second and the percentage of time spent by the application performing decision logic for inbound and outbound communication combined. In each run, message sizes were kept

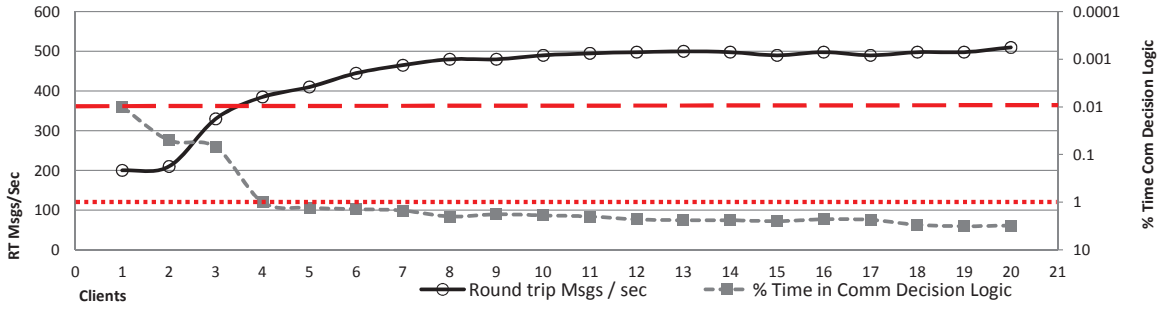

FIG. 7.6. *Server Channel, Two Servers, Non-RITA enabled, Non-Persistent.*

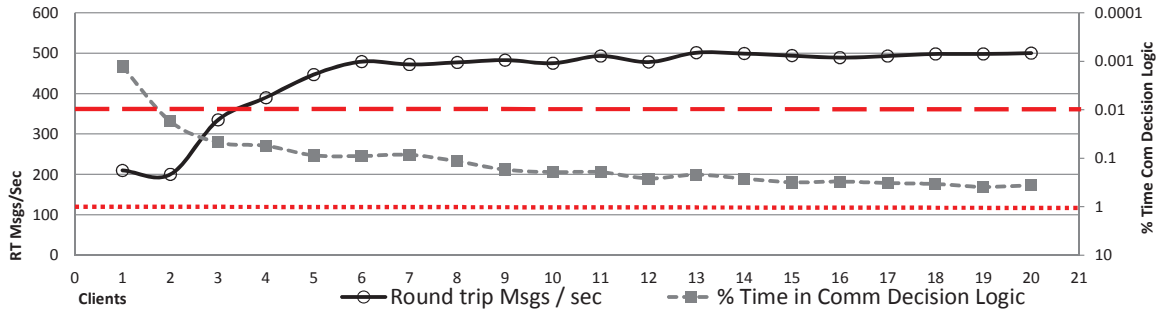

FIG. 7.7. *Server Channel, Two Servers, RITA enabled, Non-Persistent.*

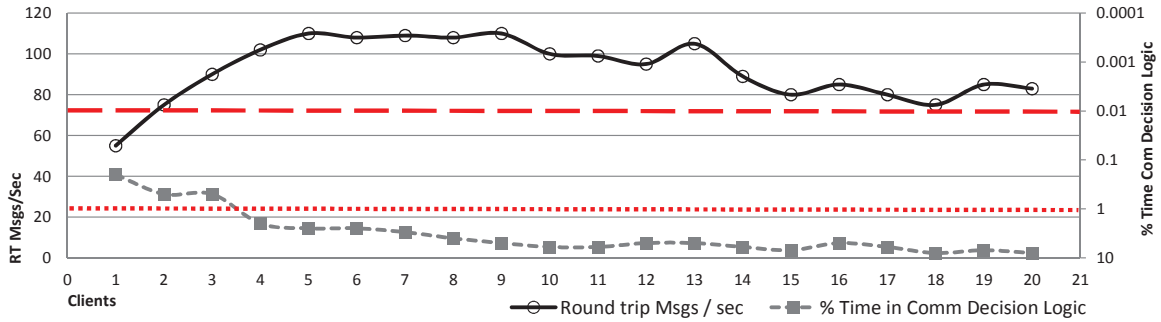

FIG. 7.8. *Server Channel, Two Servers, Non-RITA enabled, Persistent.*

constant at 262,144 bytes. Non-persistent messages were in memory without disk I/O. Persistent messages were written to disk using the MQ logging capability.

Each measurement graph has two vertical axes. The left axis shows the messages per second with values increasing vertically. The right axis shows the percent of time spent by the applications in communication decision logic — determining if the IPC is of significance to the calculations — with the values *decreasing* vertically. The red lines are the control chart values for operational limits for round-trip communication time (1.0 to 0.01 second).

7.2. Cloud PaaS with RITA. From hundreds of PaaS systems [8] [9] there are several open source systems consistently described as some of best systems to use. The PaaS systems listed are:

- AppScale (`http://www.appscale.com`)

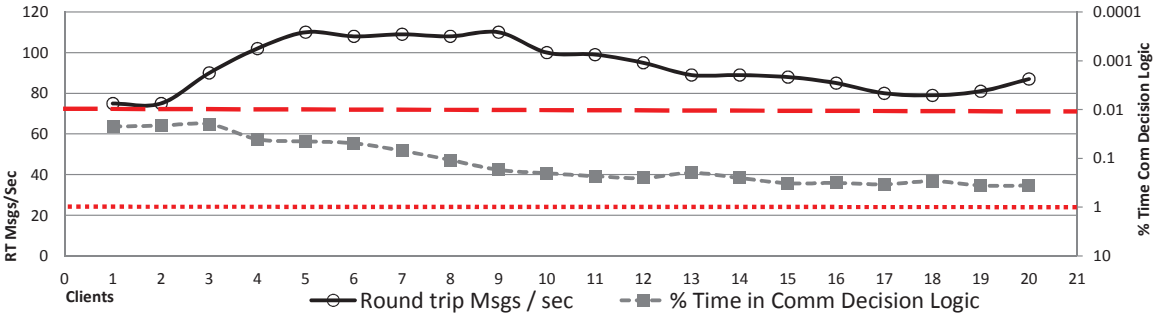

FIG. 7.9. *Server Channel, Two Servers, RITA enabled, Persistent.*

- Cloud Foundary & BOSH (`http://www.cloudfoundry.com`)
- OpenShift (`http://openshift.github.io`)

We have choosen OpenShift Origin™as the PaaS framework. OpenShift Origin/OpenStack terminology is unique to this product. Table 7.1 provides the terminology translation.

TABLE 7.1
OpenShift terminology translation to normal terminology

OpenShift Term	Translation to Normal Terminology
Broker	Host manager, controls nodes; typically a VM
Cartridge	A technology stack or framework (PHP, Perl, JEE, Python, MySQL, et.al.) to build applications
Plugin	system utilities used with a kernel; `authconfig`, DNS, et.al.
Gear	Allocation of memory, compute, and storage resources to run applications
Node	A computer; single-board, blade, et.al. Usually has only local storage for the kernel
Application	Instantiation of a cartridge (over-loaded term). Differentiated from a nominal cartridge in that a user has written additional application code that uses cartridges
Scaled App	Application instantiated in multiple gears

For OpenShift Origin, a "Gear" is a shell on a node in a shared-nothing instance of the OpenStack IaaS. The usage is to "spin-up" another "Gear" when more instances are created by the OpenShift load balance utility, HA-Proxy. Gears are run as a node O/S on VMs. A "Cartridge" is a software component that is directly used in a gear. Utilities, development tools, and database systems can be Cartridges. A Cartridge uses a specific directory template allowing it to be installed on the Gears.

Initial communication establishing a Gear in OpenStack Origin is done via `ssh` to the Broker which configures the nodes. The Broker manages multiple nodes, with each node supporting a host operating system [24]. The details of OpenShift Origin management is available from the product guides at `http://openshift.github.io/`.

Using the OpenShift Origin terms, the mapping of the RITA infrastructure to OpenShift is shown in Table 7.2. While there is no VM support in RITA, it can be deployed as a Cartridge in an OpenShift Origin PaaS. The required IPC system in OpenShift Origin is ActiveMQ which is accepted by cloud providers as a highly efficient JMS compliant system that can be tuned for the IaaS being used. Deploying RITA to OpenShift Origin is done according to the OpenShift Origin Cartridge Developer's Guide. The file format needed for the RITA OpenShift Cartridge file is detailed in the OpenShift Developers guide appendix.

Enabling technologies for IaaS distribution include the Distributed Management Task Force (DMTF) Open Virtualization Format (OVF) [6], the Distributed Resource Management Application API (DRMAA) [20] sponsored by the Open Grid Forum, and a derivative of DRMAA, Simple Linux Utility for Resource Management (SLURM). OVF is an open standard for packaging and distributing virtual appliances, or more generally soft-

ware, to be run in virtual machines. As part of a PaaS, RITA would be delivered as an OpenShift Cartridge file and a corresponding OpenStack virtual machine could be delivered via an OVF formatted system image file.

TABLE 7.2
OpenShift, RITA Comparison

OpenShift	**RITA**
Cartridge	Event monitor & engine
Plugin	IPC I/F
Gear	Cell
Node	System
Application	Application
Broker	No VM control in RITA

7.3. RITA PaaS Application. Wallace, Turchenko, et. al. [31] perform an analysis of spot market prices using a Multi-Layer Perceptron (MLP) model with back propagation error training. In MLP the "moving simulation mode" provides a short-term prediction with re-training allowing capture of the most recent, significant data from the previous prediction step to continually update and improve performance. These predictions rely on a single data stream and would be improved with event-based information from other data streams which affect spot prices. This prediction system requires a series of RITA systems monitoring events — with a condition-event matrix — controlling the event input (i.e. "gating") to the MLP model. The gating is dynamic, depending on events processed by RITA and is a classical RITA event system (Figure 7.1). RITA would be a specific improvement on the neural network sample "window" where older gross data makes the re-training less effective. RITA would allow only significant data to enter into the window for use by the neural network model.

8. State of the Art and Related Work. The current state of the art for DEBS is embodied by pub/sub communication systems or SQL semantics from corporations; IBM, TIBCO, and Oracle being the top three recognized. Indeed, a recent survey [4] shows that DBMS and MapReduce are the primary event processing semantics for cloud based systems. There have been several significant DEBS created from 2009 onward. Comparing and contrasting RITA with these systems is beyond the scope of this paper, but suffice it to say that the RITA condition-event matrix, canonical event types, and declarative and imperative bifurcation do not exist in any cloud or DEBS system. A contrast and comparison may be done at a later date for the DEBS that are still being maintained. For several of the DEBS documented, the interest and continued development for these systems has become dormant.

In [25] Hermes, Gryphon, Siena, Esper, Borealis and Aurora (now just Borealis), and AMIT are discussed. Esper, now a product from ExperTech[3], is currently available under GPL license [25]. Borealis is a second-generation distributed stream processing engine. Borealis inherits from Aurora [3] and from Medusa [26]. AMIT is now an IBM e-business Management Service offering.

In [17] Java Event-Based Distributed Infrastructure (JEDI), Rebeca notification service, Cambridge Event Architecture (CEA), Elvin notification service, READY event notification service, and Narada Brokering project are discussed. Also discussed are the commercial JMS pub/sub systems: IBM MQ, TIBCO, and Oracle. The CEA, JEDI, Siena, Hermes, Gryphon, Esper, RuleCore, and AMIT Research projects, and industrial solutions, work on event stream processing (ESP) and complex event processing (CEP). These two approaches address processing large amounts of events delivering real-time communication, allowing closed loop decision making, and continuous data integration [25].

In the DEBS mentioned above, each is based on the Object Management Group[4] pub/sub model with additional attributes for object oriented data types, SQL event streaming, query modification, and large database systems but *without* the novel RITA ability to reduce the actual application code processing of decision logic via a condition-event matrix that enables the ability to regulate event interaction with application code, temporally

[3]http://www.espertech.com
[4]http://www.omg.org/

handle heterogeneous (i.e. divergent) time domains between systems, and provide an formal isomorphic mapping for events and actions (i.e. $E \mapsto A$).

9. Future Work. In this paper the RITA theory, use, and mechanics have been shown to be portable from multiprocessing shared memory systems (avionics), grid systems (telecommunications), and has applicability to cloud computing systems. To improve the RITA specifications for a cloud computing environment, the implementation language for portions of the event engine and event dispatch portions of the event monitor need porting from the current C and C++ code base to a language developed for concurrency in cloud computing environments. The recently developed Go language sponsored by Google [10] provides good programming support for RITA. A re-hosting of RITA concepts in Go is advantageous as event propagation is naturally mapped to the Go language `interface` types, and Go's "Goroutines" provide a natural way to program Guards and Conditions.

Additional work would include a variant of VXDL [28]. VXDL is a prototype language for specifying virtual resources in networks. For validation of RITA specifications it can be used to describe the cloud resource instances where RITA systems can be hosted allowing an additional level of specification and modeling. This allows users to create and manage changing virtual infrastructures across the entirety of the Internet as cloud computing systems are not static. Cloud systems change component system architectures available, e.g. Amazon EC2 did not provide GPU systems, and over time did. Thus using VXDL to describe available systems allows specifications to be validated against the desired network topology and required execution time-lines. Currently RITA does not have a meta-language to map cloud computing resources. VXDL is seen as a meta-data language for assigning RITA processing cells on appropriate cloud system component systems with the needed resources for the event system execution. The VXDL meta-data can be used to define what resources are needed for a RITA schema and, instead of placing the system by hand, the VXDL meta-data can be used to automate placement of the RITA systems.

REFERENCES

[1] K. Birman, *Network perspective*, in Guide to Reliable Distributed Systems, Texts in Computer Science, Springer London, 2012, pp. 101–143.

[2] M. Boniface, B. Nasser, J. Papay, S.C. Phillips, A. Servin, Y. Xiaoyu, Z. Zlatev, S.V. Gogouvitis, G. Katsaros, K. Konstanteli, G. Kousiouris, A. Menychtas, D. Kyriazis, *Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds*, Fifth International Conference on Internet and Web Applications and Services (ICIW), 2010 , vol., no., pp.155,160, 9-15 May 2010. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5476775&isnumber=5476480. URL visited 2014-06-09.

[3] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik, *Monitoring streams - a new class of data management applications.*, in VLDB, Morgan Kaufmann, pp. 215–226.

[4] O.M. de Carvalho, E. Roloff, P.O.A. Navaux, *A Survey of the State-of-the-art in Event Processing*, proceedings of 2013 Parallel and Distributed Processing Workshop, Institute of Informatics, Federal University of Rio Grande do Sul, Brazil. http://inf.ufrgs.br/gppd/wsppd/2013/papers/wsppd2013_submission_14.pdf.mod.pdf URL visited 2014-06-09

[5] N. Deakin, *JSR-000914 Java Message Service (JMS) API*. http://jcp.org/aboutJava/community/process/final/jsr914/. URL visited 2013-08-08.

[6] Distributed Management Task Force, *Open Virtualization Format Specification*, http://dmtf.org/standards/ovf URL visited 2013-10-01.

[7] I. Fedotova, E. Siemens, and H. Hu, *A high-precision time handling library*, in ICNS 2013 : The Ninth International Conference on Networking and Services, IARIA, 3 2013, pp. 193–199.

[8] FindTheBest, *Compare Cloud Computing Providers*. http://cloud-computing.findthebest.com URL visited 2013-08-18.

[9] Google,*Cloud Computing Providers*, http://en.wikipedia.org/wiki/Category:Cloud_computing_providers URL visited 2013-08-18.

[10] R. Griesemer, R. Pike, and K. Thompson, *Go language*. http://www.golang.org/. URL visited 2013-09-07.

[11] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[12] IBM, *WebSphere MQ*. http://www-03.ibm.com/software/products/us/en/wmq. URL visited 2013-08-08.

[13] IEEE, *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002), (2009). doi: 10.1109/IEEESTD.2009. 4772740.

[14] Intel, *IA-PC HPET (High Precision Event Timers) specification*, tech. report, Intel, 10 2004.

[15] F. Kröger and S. Merz, *Temporal Logic and State Systems. Texts in Theoretical Computer Science. An EATCS Series*, Springer, 2008.

[16] J. E. McDonald, *A network architecture for data-driven systems*, Tech. Report ADA168764, AIR FORCE WRIGHT AERONAUTICAL LABS WRIGHT-PATTERSON AFB OH, 1985.

[17] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed event-based systems*, vol. 1, Springer Heidelberg, 2006.
[18] Object Management Group, *Common Object Request Broker Architecture (CORBA)*. http://www.omg.org/ spec/CORBA/. URL visited 2013-08-08.
[19] Object Management Group, *OMG specifications*. http://www.omg.org/spec/#MW. URL visited 2013-09-01.
[20] Open Grid Fourm, *Distributed Resource Management Application API Version 2 (DRMAA)*, http://www.gridforum.org/ standards/ URL visited 2013-10-01
[21] Oracle, *Oracle Tuxedo*. http://www.oracle.com/us/products/middleware/cloud-app-foundation/tuxedo/message-queue/ overview/index.html. URL visited 2013-08-08.
[22] A. Paschke and P. Vincent, *A reference architecture for event processing*, in Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09, New York, NY, USA, 2009, ACM, pp. 25:1–25:4.
[23] J. Poole, *MCI and BT take on telcom thieves*, InfoWorld, vol. 19, issue 37, pp. 63, Sep., 1997.
[24] Red Hat, *OpenShift Origin System Architecture Guide*, http://openshift.github.io/documentation/oo_system_architecture_ guide.html URL visited 2013-09-01.
[25] S. Rozsnyai, J. Schiefer, and A. Schatten, *Concepts and models for typing events for event-based systems*, in Proceedings of the 2007 inaugural international conference on Distributed event-based systems, DEBS '07, New York, NY, USA, 2007, ACM, pp. 62–70.
[26] S. Z. Sbz, S. Zdonik, M. Stonebraker, M. Cherniack, U. C. Etintemel, M. Balazinska, and H. Balakrishnan, *The aurora and medusa projects*, IEEE Data Engineering Bulletin, 26 (2003).
[27] TIBCO, *TIBCO rendezvous*. http://www.tibco.com/products/automation/messaging/high-performance-messaging/ rendezvous/default.jsp. URL visited 2013-08-08.
[28] P. Vicat-Blanc Primet, T. Kudoh, and J. Mambretti, *VXDL: Virtual resources and interconnection networks description language*, in Networks for Grid Applications, vol. 2 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, 2009, pp. 138–154.
[29] R. M. Wallace, *Regulated isomorphic temporal architecture*. http://emendo-ex-erratum.org/papers/RITA.pdf. URL visited 2013-09-07.
[30] R. M. Wallace, J. E. McDonald, and D. O. Hague, *A data-driven operating system for data-driven architectures of real-time systems*. http://emendo-ex-erratum.org/papers/RMW_DH_JM_DDOS-DDA.zip. URL visited 2013-09-07.
[31] R. M. Wallace, V. Turchenko, M. Sheikhalishahi, I. Turchenko, V. Shults, J. Vazquez-Poletti, and G. L., *Applications of neural-based spot market prediction for cloud computing*, in International Conference on Intelligent Data Acquisition and Advanced Computing Systems IDAACS 2013, vol. 2, IEEE, 2013, pp. 710–716.

Edited by: Dana Petcu
Received: Apr 14, 2014
Accepted: Jun 24, 2014

DOI 10.12694/scpe.v15i2.983
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 15, Number 2, pp. 187–200. http://www.scpe.org

GENERALIZED MATRIX MULTIPLICATION AND ITS OBJECT ORIENTED MODEL

MARIA GANZHA[1], MARCIN PAPRZYCKI[2], AND STANISLAV G. SEDUKHIN[3]

Abstract.
Since the beginning of the 21st century, we observe rapid changes in the area of, broadly understood, computational sciences. One of interesting effects of these changes is the need for reevaluation of the role of dense matrix multiplication. The aim of this paper is two-fold. First, to summarize developments that point toward a need for reconsidering usefulness of matrix multiplication generalized on the basis of the theory of algebraic semirings. Second, to propose generalized matrix-matrix multiply-and-update (MMU) operation and its object oriented model.

Key words: matrix multiplication, algebraic semirings, algebraic path problem

AMS subject classifications. 65F30, 13A99

1. Introduction. Recently, a number of changes can be observed in computational sciences. They concern all levels of the computational stack. First, evolution of computer hardware, forced by limits imposed by physics, resulted in practical disappearance of processors with a single computational unit. As a matter of fact, today it is possible to have a quad-core processor in a cell phone (e.g. in the newest Samsung Galaxy 4) and even 8 cores (in the Motorola X8 Mobile Computing System [10]). Furthermore, it is already possible to have more than a thousand fused multiply and add (FMA) units in a single GPU processor [33]. Second, there is a constantly growing gap between the capacity of the processor to consume the data and hardware' ability to feed it. Third, rapidly decreasing cost of the FMA unit, combined with appearance of processors with thousands FMAs, lead to suggestions that a complete reevaluation of approach to computing is needed [34, 35]. Here, the basic assumption is that data access/movement is "expensive," while arithmetic operations are "cheap." Fourth, it is time to (re)consider complexity of codes that try to match (and effectively utilize) current computer hardware with as much as seven levels of data access latency. Finally, rapid proliferation of devices with matrix-like sensor input (e.g. digital cameras, medical imaging devices, radio telescopes, etc.) forcefully reminds us that, in multiple applications, actual data consists of 2D and/or 3D matrix-structures that are fed with high speed, and should not be stored but processed in-place as they elements are delivered to the processing units.

In this paper we will argue that the time has come for a meta-reflection and general change of approach to large-scale (primarily "scientific") computing. In particular, it is important to look into efficient solution of matrix-based problems, and this is precisely the scope of the current contribution. This paper modifies and extends our two conference papers [69, 30], and it is organized as follows. First, we discuss the interaction between progress in computer hardware and computational linear algebra in the early days of supercomputing. Second, we consider dense matrix multiplication, as one of key elements of large number of linear algebraic algorithms. Here, we also look into its generalization through the theory of algebraic semirings. Next, we reflect on the most current trends in hardware and software for computational linear algebra and combine these considerations to propose a generalized matrix multiply and update (MMU) operation. Finally, we use the discussion of the state-of-the-art in object oriented BLAS to propose an object oriented realization of the generalized MMU.

2. Computer hardware and computational linear algebra in the long gone past.

2.1. Single-processor computers. Let us start our discussion from late 1970's, when it became clear that many algorithms for matrix computations consist of similar building blocks (e.g. a *vector update*, or a *dot-product*). As a result, in the Cray-1 supercomputer, vector operation $\bar{y} \leftarrow \bar{y} + \alpha\bar{x}$ (where $\bar{x}$ and $\bar{y}$ are

[1]Systems Research Institute, Polish Academy of Sciences, Warsaw and Institute of Informatics, University of Gdańsk, Gdańsk, Poland (Maria.Ganzha@ibspan.waw.pl),
[2]Systems Research Institute, Polish Academy of Sciences, Warsaw and Department of Management and Technical Sciences, Warsaw Management Academy, Warsaw, Poland (Marcin Paprzycki@ibspan.waw.pl),
[3]Distributed Parallel Processing Laboratory, The University of Aizu, Aizuwakamatsu City, Fukushima 965-8580, Japan (sedukhin@u-aizu.ac.jp)

n-element vectors, while α is a scalar) has been efficiently implemented. Specifically, S. Cray proposed vector processors with *chaining* of multiply and update operations [66], where results of the multiply operations have been forwarded directly from the "multiplication unit" to the "addition unit." A few years later, the IBM build the 3090 series of vector computers with efficient implementation of the *dot-product* operation [40]. In the meantime, in 1979, the level 1 BLAS standard was proposed [44], which defined a set of core vector-vector operations. The assumption behind the level 1 BLAS was that the computer vendors would provide efficient hardware (and software) realizations of these operations. In this spirit, the Cray Inc. build computers with efficient *vector updates*, that became a part of the *scilib* library [39]; while the IBM developed the $ESSL$ library [38], with highly optimized dot-products. This library was later ported to the IBM RS/6000 workstations; the first commercial computer to implement the fused multiply-and-add (FMA) operation [52]. Note that the FMA operation appears in both vector updates and dot products. Following this path, currently, processors from IBM, Intel, AMD, Nvidia, and others [18, 20, 24, 41, 53, 55, 56], include efficient hardware supported scalar floating-point fused multiply-add operation.

The FMA combines two basic floating-point operations (flops) into one (three-read-one-write) operation with only one rounding error, throughput of two flops per cycle, and a few cycles latency – depending on the depth of the FMA pipeline. There exist two FMA standards: FMA3 and FMA4. The difference is that the FMA4 has four operands ($d \leftarrow c + a * b$), while the FMA3 has three operands (it is an update $c \leftarrow c + a * b$). Since it is not our aim to discuss the philosophical differences between these two approaches, for the purpose of this paper, let us assume that the term "FMA" covers both standards. Besides the increased accuracy, the FMA minimizes operation latency, reduces hardware cost, and chip busing [52]. The standard floating-point add, or multiply, are performed by taking $a \leftarrow 1.0$ (or $b \leftarrow 1.0$) for addition, or $c \leftarrow 0.0$ for multiplication. Therefore, the two floating-point constants, 0.0 and 1.0, need to be available within the processor. As a result, in the current computer hardware, one cannot "practically distinguish" between (faster/simpler) addition and (slower/more complex) multiplication. This fact has important consequences for some classes of divide-and-conquer algorithms (see, below).

The development of software for computational linear algebra continued, with introduction of level 2 BLAS, standardizing matrix-vector operations [28]. Next, the level 3 BLAS was introduced in [27], defining matrix-matrix operations. Following, the supercomputer vendors developed highly optimized implementations of BLAS kernels for their hardware (e.g. the Cray Inc. provided Cray Assembly Language based implementations [39]). Furthermore, on the basis of the level 3 BLAS, the LAPACK library was proposed [14], defining templates for solving dense matrix problems.

It is also during these years, when the disparity between the speed of the processor and the memory access time became apparent (see, for instance [17]). In response to this trend, computers with hierarchical memory (introduced for data reuse) have been proposed. To match these architectures, block-oriented realizations of fundamental linear algebra algorithms for dense matrices have been introduced and experimented with (this is precisely the class of algorithms that constituted the core of the "LAPACK approach"). Here, blocking allowed data reuse and minimization of data movement. Since the process of finding perfect blocking was rather tedious, and only few programmers were ready to do this for the highly complex computer hardware (see, for instance, [31]), auto-tuners have been introduced. Among them, the ATLAS project [76] became the most popular.

2.2. Parallel computers. Let us now look into historical trends in *parallel computer hardware*. In the 1990's three main designs for the parallel computers were: (1) array processors, (2) shared memory parallel computers, and (3) distributed memory parallel computers. While quite popular initially, array processor supercomputers disappeared by approximately 1995. The main reason was an apparent lack of flexibility to deal with problems that do not natively appear in the "matrix-form." Furthermore, it is worthy observing that, regardless of the array organization of processors, no truly efficient implementation of matrix multiplication has been realized. Next, the shared memory parallel supercomputers started to fade away. Here, the main problem was the bottleneck caused by the connection between the memory and the processors. It turned out that, in computational practice, scaling such machines to more than 32 processors was extremely difficult and economically unfeasible. As the result, by the end of 20th century, the dominating supercomputer architecture became the distributed memory parallel computers.

As what concerns computational linear algebra, while the LAPACK library was focused on single-processor and shared-memory parallel computers, the ScaLAPACK [21] was to provide the same standardization for the distributed memory computers. The ScaLAPACK is based on the single program multiple data (SPMD) programming model. In addition to the computational kernels, parallel BLAS kernels [22], and communication routines (BLACS [5]) have been defined. Overall, the main assumption remained that the hardware/software vendors are going to provide efficient low-level realization of the needed functionality. Interestingly, while the LAPACK project became quite successful, the ScaLAPACK did not match its reach and popularity. There exist multiple of reasons for this fact. Among others: (1) programming distributed memory computers turned out to be more difficult than expected, (2) end of the Cold War cut funding for development of hardware and software on the large scale, (3) raise of cluster (COTS) computers moved the interest to low-cost solutions on the small scale (where highly optimized routines were not the biggest concern), (4) extending / updating Fortran 77 turned out to be a complete failure, (5) implementers of codes for computationally intensive problems turned to C and next to C++ and object oriented scientific computing (however, object orientation did not result in unification of the field; see, section 7).

In summary, there was a time when development of "high performance" hardware and software worked hand-in-hand. However, their pathways have diverged and software developers have been left to catch up with the computer hardware. Nevertheless, matrix multiplication remained the workhorse of a very large class of algorithms in computational linear algebra (in particular, their block-oriented implementations). Therefore, we will now focus on high performance matrix multiplication.

3. Dense matrix multiplication in scientific computing. While dense matrix-matrix multiplication appears in many computational problems, its most popular application is in the update step of block algorithms and has the form: $C \leftarrow C + A \times B$ (where A, B and C are appropriately dimensioned matrices; note that, in general, A and /or B can be in transposed form, which leads to four different variants of this operation). Despite its simplicity, the arithmetic complexity, and data dependencies, make it a challenging problem to reduce the *run-time complexity*. The two basic approaches to speeding matrix multiplication were, first, lowering the *arithmetic* complexity – achieved by reducing the number of scalar multiplications (complex/expensive), while increasing the number of scalar additions/subtractions (simple/cheap). Here, one could list Strassen [73], Pan [57], and Coppersmith-Winograd [23] algorithms (further discussion and references can be found in [65]). Second, by the *parallel implementation* [13, 75, 42, 29, 19, 47]. Of course, a combination of these two approaches is also possible (see, for instance, [72, 37, 32, 49]).

The recursive matrix multiplication "worked well" from the point of view of theoretical analysis of arithmetical complexity, and when implemented on early computers. However, its implementation started to became a problem on computers with hierarchical memory (e.g. to reach optimal performance of a Strassen-type algorithm, recursion had to be stopped when the size of the divided submatrices approximated the size of the cache memory – differing between machines; see, [15]). Furthermore, practical implementation of Strassen-type algorithms requires extra memory (e.g. the Cray's implementation of Strassen's algorithm required extra space of order $2.34n^2$ [25]). Here, recall two facts. First that in the modern FMA units, there is no distinction between time of addition and multiplication (these two operations cannot be "separated" into cheap and expensive). Second that the speed of memory access is one of the key factors limiting advances of high performance computing.

It is worthy recalling that recursive matrix multiplication has been successfully used within other algorithms, e.g. within blocked Gaussian Elimination [15, 61, 60, 58]. Unfortunately, recursive approaches result in problems with numerical stability. While, as proved by N. J. Higham in [36], these problems should not be extremely pronounced, in [61] it was shown that they may prevent solution of at least some practical problems. Specifically, for a class of PDE problems, substituting matrix multiplication by a fast one, in the block-oriented linear equation solver, resulted in a failure to reach the solution.

The second approach to speeding matrix multiplication is through the design of parallel algorithms, which have been implicitly or explicitly based on a time-space scheduling of FMAs in the 3D computational index space. This scheduling directly affects data reuse, by orchestrating data movement between different scalar operations. One of the key differences between this and the recursion-based approaches is that in the parallelized matrix multiplication, patterns of data movement remain relatively simple and well structured. Moreover, in

the recent paper [70], it was shown that, actually, there exist only four basic classes of schedules and, therefore, classes of algorithms. In other words, **all** existing parallel matrix multiplication algorithms are extensions of these four basic schedulings, with respect to different matrix shapes, blocking or tiling, dimensionality of parallel implementation, underlined (assumed) computer architecture, etc. (see, also [46]). Furthermore, it was observed that all four schedulings can be used to implement the level 3 BLAS operation `_GEMM` [27] of the form $C \leftarrow C + A \times B$; the *matrix multiply-update* (*MMU*). Out of the four classes of parallel *MMU* algorithms defined in [70]: (i) Broadcast-Compute-Shift; (ii) All-Shift-Compute (or Systolic); (iii) Broadcast-Compute-Roll; and (iv) Compute-Roll-All (or Orbital), the last one is characterized by regularity and locality of data movement, maximal data reuse without data replication, recurrent ability to involve into computing all matrix data at once (focal-plane I/O), etc. This makes it well suited for the computer hardware that is likely to materialize in the near future (see, Section 5). Interestingly, the recently proposed 2.5D approach to matrix multiplication [71], represents a hybrid between a Broadcast-Broadcast-Compute (BBC) and a Compute-Roll-All (CRA). Specifically, when no extra memory is available the approach reduces to the CRA (Cannon Algorithm), while when N extra copies of appropriate matrices (where N is the size of the, square, matrices involved in multiplication) can be stored in the system, the matrix multiplication operation becomes a version of the Broadcast-Compute-Roll. However, an in-depth discussion of this point is out of scope of this contribution.

Obviously, it is possible to combine recursive and parallel approaches to dense matrix multiplication. Here, the situation becomes even more complex, as irregularity of data movement is exaggerated through the complexity of the underlying hardware (e.g. extra levels of latency of data access). However, with a lot of programmer's work, hybrid approaches outperformed the standard parallel matrix multiplication [72, 37, 32, 59]. Interestingly, the most recent research seems to contain two contradictory claims. First, results presented in [26] support the conclusion that Strassen-type approaches are not practically beneficial on current computer architectures. Second, in [49], benefits of the 2.5D Strassen multiplication are praised. However, careful study of results presented there (see, [49], Figure 1.f) indicates that the apparent performance gain (reported in, [49], Figure 1.e) is purely virtual, as it does not result in substantial reduction of the wall-clock time. This could be used as an argument supporting the conclusions reported in [26].

Let us leave the discussion concerning advantages and disadvantages of specific implementations of parallel dense matrix multiplication and observe that this operation appears also in a much broader context. Specifically, matrix multiplication can be generalized through the theory of algebraic semirings. Let us now look into this topic in more detail.

4. Algebraic semirings in scientific calculations. Since 1970's, a large number of problems has been combined under a single umbrella, named the *Algebraic Path Problem* (*APP*; see [45, 16, 67]). Furthermore, it was established that the matrix "multiply-and-update" (MMU) operations, in different algebraic semirings, can be used as a centerpiece of various APP solvers.

Let us start from the needed definitions. A closed (scalar) semiring $(S, \oplus, \otimes, *, \bar{0}, \bar{1})$ is an algebraic structure defined for a set S, with two binary operations: addition $\oplus : S \times S \to S$ and multiplication $\otimes : S \times S \to S$, a unary operation called *closure* $\circledast : S \to S$, and two constants $\bar{0}$ and $\bar{1}$ in S. Here, we are particularly interested in the set S consisting of matrices. Thus, following [45], we introduce a matrix semiring $(S^{n\times n}, \bigoplus, \bigotimes, \bigstar, \bar{O}, \bar{I})$ as a set of $n \times n$ matrices $S^{n\times n}$ over a closed scalar semiring $(S, \oplus, \otimes, *, \bar{0}, \bar{1})$ with two binary operations, matrix addition $\bigoplus : S^{n\times n} \times S^{n\times n} \to S^{n\times n}$ and matrix multiplication $\bigotimes : S^{n\times n} \times S^{n\times n} \to S^{n\times n}$, a unary operation called *closure of a matrix* $\bigstar : S^{n\times n} \to S^{n\times n}$, the zero $n \times n$ matrix $\bar{O}$ whose all elements equal to $\bar{0}$, and the $n \times n$ identity matrix $\bar{I}$ whose all main diagonal elements equal to $\bar{1}$ and $\bar{0}$ otherwise. Here, matrix addition and multiplication are defined as usually in the linear algebra.

As stated, large number of matrix semirings appear in well-studied APPs. We summarize some of them in a table (similar to that presented in [12]). For simplicity of notation, in the Table 4.1, we represent them in a scalar form.

Note that the *Minimum reliability path* problem has not been encountered by the authors before. It was defined on the basis of systematically representing possible semirings—as a natural counterpart to the *Maximum reliability problem* (the only difference is the $\oplus$ operation: *min* instead of *max*). Since the maximum reliability path defines the *best* way to travel between two vertices of a graph; the *Minimum reliability problem* could be interpreted as: finding the *worst* pathway, one that should not be "stepped into").

TABLE 4.1
Semirings for various APP problems.

S	$\oplus$	$\otimes$	$\circledast$	$\bar{0}$	$\bar{1}$	Application
0,1	$\vee$	$\wedge$	1	0	1	Reflexive and transitive closure of binary relations
$\mathbb{R}$	$+$	$\times$	$1/(1-r)$	0	1	Matrix inversion
$\mathbb{R}_+ \cup +\infty$	min	$+$	0	∞	0	All-pairs shortest paths
$\mathbb{R}_+ \cup +\infty, -\infty$	max	$+$	0	-∞	0	Maximum cost (critical path)
$[0,1]$	max	$\times$	1	0	1	Maximum reliability paths
$[0,1]$	min	$\times$	1	0	1	Minimum reliability paths
$\mathbb{R}_+ \cup +\infty$	min	max	0	∞	0	Minimum spanning tree
$\mathbb{R}_+ \cup -\infty$	max	min	0	-∞	0	Maximum capacity paths

While Table 4.1 summarizes the *scalar* semirings, and *scalar* "multiply-and-add" operations, kernels of blocked algorithms for solving the APP, are based on (block) *matrix* "multiply-and-update" (MMU) operations [74, 51]. Therefore, let us present the relation between the *scalar* (fused) "multiply-and-update" (FMA) operation (ω), and the corresponding *matrix* "multipy-and-update" (MMU) kernel (α), for semirings in Table 4.1 (here, Nb is the size of a matrix block; see, also [68]):

- Matrix Inversion Problem (MIP):
  (α) $a(i,j) \leftarrow a(i,j) + \sum_{k=0}^{Nb-1} a(i,k) * a(k,j)$;
  (ω) $c \leftarrow a \times b + c$;
- Shortest Paths Problem (SPP):
  (α) $a(i,j) \leftarrow \min\big(a(i,j), \min_{k=0}^{Nb-1}[a(i,k) + a(k,j)]\big)$;
  (ω) $c \leftarrow \min(c, a+b)$;
- Critical Path Problem (CRP):
  (α) $a(i,j) \leftarrow \max\big(a(i,j), \max_{k=0}^{Nb-1}[a(i,k) + a(k,j)]\big)$;
  (ω) $c \leftarrow \max(c, a+b)$;
- Maximum Capacity Paths Problem (MCP):
  (α) $a(i,j) \leftarrow \max\big(a(i,j), \max_{k=0}^{Nb-1} \min[a(i,k), a(k,j)]\big)$;
  (ω) $c \leftarrow \max[c, \min(a,b)]$;
- Maximum Reliability Paths Problem (MaRP):
  (α) $a(i,j) \leftarrow \max\big(a(i,j), \max_{k=0}^{Nb-1}[a(i,k) \times a(k,j)]\big)$;
  (ω) $c \leftarrow \max(c, a \times b)$;
- Minimum Reliability Paths Problem (MiRP):
  (α) $a(i,j) \leftarrow \min\big(a(i,j), \min_{k=0}^{Nb-1}[a(i,k) \times a(k,j)]\big)$;
  (ω) $c \leftarrow \min(c, a \times b)$;
- Minimum Spanning Tree Problem (MST):
  (α) $a(i,j) \leftarrow \min\big(a(i,j), \min_{k=0}^{Nb-1}[\max\big(a(i,k), a(k,j)\big)]\big)$;
  (ω) $c \leftarrow \min[c, \max(a,b)]$.

Summarizing, application of algebraic theory of semirings allows bringing together multiple problems, solution of which involves generalized matrix multiplication. Let us now look how this fits with current and predictable trends in computational sciences, in particular in computational linear algebra.

5. Current trends in high performance computing. Recently, high performance computer hardware found itself "in a box" imposed by physics. As the "floor" we can see the limits on miniaturization. It is physically impossible to continue reducing the size of the processor without having to deal with quantum effects. On the "ceiling" we have the heat dissipation problem. The current on-chip processor designs make it extremely difficult to cool the densely packed (miniaturized) transistors. Finally, the memory "walls" make it increasingly complex to feed the data-starving processors. Here, the use of hierarchical memory with up to three levels of cache memory provided only a temporary solution, with a price in workload of software implementers (see, also, below). An obvious solution to these problems is to combine multiple computational units. However, here the "speed of light problem" starts to materialize again. To keep the system synchronous, it is possible to send signal only for a certain distance (within a single clock cycle). This effect can be seen not only when

considering design of systems with billions of processors that fill a room of a size of a football stadium. We can observe it also within multi-core chips. For instance, the frequency of Pentium processors reached 3.8 GHz (Pentium 4, Prescott), while the Nvidia Tesla processor runs at about 700 MHz. Here, through the reduction of frequency, it is possible to keep the multiple cores synchronous.

In this way, we have pointed to two recent trends in design of high performance hardware. First, today's multi-core processors can be seen as shared memory parallel computers on-chip. Second, GPU chips represent ideas originating from the SIMD supercomputers of the old. Both trends feature multiple FMA units (from a few FMAs in multi-core processors, up to more than a thousand in the newest GPUs). They also follow two different roadmaps to the exascale computing. One of them involves smaller number of more powerful (more complex) processors, while the other is based on larger number of less powerful (simpler) processing units. In a way, high performance computing has spiraled back to the 1990's. For instance, in 1993, the MassPar MP-2216 array processor with 2048 "cores" took 180's position at the TOP500 computer list [9]. It was capable of delivering 2.4 GFlops. Today's Tesla K20X GPU, produced by Nvidia, has 896 double precision FMAs (2688 single precision FMAs that work in triples to deliver double precision operations) and is capable of delivering 1.31 TFlops [33]. In other words, a TOP200 computer from 1993 was about 500 times slower than a single (multi)processor of today, while both had a comparable number of computational units.

As what concerns software, some researchers work on squeezing performance out of existing computer architectures (see, for instance, recent results from the team led by J. Demmel [49, 71]). Others see the future of computing in proposing complex kernels that are to replace the BLAS standard (see, [80]). However, it is also possible to see the problem from the perspective of William of Ockham, who suggested that "one should proceed to simpler theories." Taking into account what has been said so far, it should be clear that *simplicity and uniformity of data manipulation* should become the driving principle behind exa/zetta-scale computer system design. Here, the work of Sedukhin et.al. [70] focuses on development of a novel computer architecture capable, among others, of delivering efficient parallel matrix multiplication. In the same context, for sparse matrix operations, John Gustafson stated: "Go Ahead, Multiply by Zero!" [34]. His assumption is that in the hardware of the future, cost of arithmetic operations will be so low, in comparison with data movement (and indexing), that fundamental assumptions behind current approaches to computational sparse (and dense) linear algebra will have to be re-evaluated. Interestingly, the team of J. Dongarra has recently initiated a new project, in which they investigate possibility of reintroduction of systolic architectures to the computing mainstream [43]. Here, it is also worthy envisioning that this could mean that we may, some time in the future, forget about rectangular matrices (in computational practice). Instead we will pad them with zeros to make them square and in this way match the square arrangements of FMA's within processors. Since the price of an FMA is currently (June 2013) at about 22 US cents and dropping (see, [77]), such approach does not seem so unreasonable; especially, taking into account the cost of data manipulations involved in dealing with rectangular structures. This should be kept in mind when considering the fact that the APP algorithms (mentioned above) involve square matrices only.

Let us present a few more examples of a slow shift in view on programming modern (and future) high-performance computers (which are expected to be build of hundreds of thousands of processors with thousands of FMA units each). First, initial results concerning an implementation of the 2D FFT (2048×2048)-point, on a 24-wide FMA Cell/B.E.@PS3 processor shows only a moderate difference ($\sim$3 times) in performance over non-recursive matrix multiply and add-based 2D DFT, despite almost 30 times difference in the number of arithmetic operations [79]. Moreover, it was recently reported (in [48]) that, by using two quad-core Intel Nehalem CPUs, the direct convolution approach outperforms the FFT-based approach by at least five times, even when the associated arithmetic operation count is approximately two times higher. This "imbalance" towards non-recursive approach is directly related to efficient use of multicore processors achieved through efficient dense MMU implementation.

Second, the early work in sparse linear algebra was guided by the desperate need of saving memory. As a result, a number of "compressed matrix formats" has been proposed and experimented with. However, in the most recent work, instead of dealing with individual matrix entries, the basic element becomes a "dense block" (of size related to the processor cache; see, for instance [78, 50] and references collected there). Here, we observe the same general pattern as above, where simplicity of data access pattern(s) compensates for the

higher operation count caused by multiplication by zero (note that some of such blocks may have only a few non-zero elements).

Separately, the work reported in [50] illustrates one more important aspect of using highly optimized, memory-preserving schema to deal with sparse matrix multiplication. The code generator that allows to deal with various forms of sparse matrix-vector multiplication, for various data types and matrix formats is approximately 6,000 lines long. However, the generated code is more than 100,000 lines long. This code can efficiently work on hierarchical memory multicore systems with up to 16 cores (see, [50]), but will require further tuning for larger number of FMAs (and it is neither GPU nor distributed memory computers oriented). This being the case, when thinking about fast matrix / matrix-vector multiplication, one needs to consider also the programming effort required to develop and later modify (re-optimize) codes based on complex data structures and movements. This observation provides also context for the question of long-term feasibility of approaches similar to the GOTOBLAS [31]. Furthermore, the original direction of the BLAS project needs to be considered. Part of the success of the BLAS was related to its simplicity (including relatively small number of basic routines). Therefore, it is not clear if proposing a standard with many complex kernels (as in [80]) is going to be successful in a long run.

Finally, let us look into relation between the APP and the recent trends in computing. While algebraic semirings can be seen as a simple "unification through generalization" of a large class of computational problems, they should be viewed in the context of, mentioned above, success of fused multiply-and-add (FMA) units. Note for instance that, the GPU processors from Nvidia and AMD combine large number of FMA units (e.g. the Nvidia's Tesla chip allows 2688 single-precision FMA operations completed in a single clock cycle [33]). In this context, recent experiments show that FMA-based kernels speed-up ($\sim 2\times$) solution of many scientific, engineering, and multimedia problems, which are based on matrix transforms [35]. However, other APPs "suffer" from lack of hardware support for the needed scalar FMA operations (e.g. those listed in Table 4.1). The need for min or/and max operations introduces one or two conditional branches, or comparison/selection instructions, which are highly undesirable for deeply pipelined processors. Recall that each of these operations is repeated Nb-times in the corresponding kernel (see, Section 4), while the kernel itself is called many times in the blocked APP algorithm. Here, note the recent results, reported in [68], which involve evaluation of an MMU operation in different semirings, on the Cell/B.E. processor. They showed that the "penalty" for lack of a *generalized* FMA unit may be up to 400%. This can be also interpreted as follows: having an FMA unit, capable of supporting operations and special elements from Table 4.1 could speed-up solution of APP problems by up to 4 times. Interestingly, we have just found that the AMD Cypress GPU processor supports the combined scalar $(\min, \max)$-operation through a single call with only 2 clock cycles per result. In this case, the Minimum Spanning Tree (MSP) problem (see, Table 4.1) could be solved more efficiently than previously realized. Furthermore, this could mean that the AMD hardware has $-\infty$ and ∞ constants already build-in. This, in turn, could constitute an important step towards hardware support of generalized FMA operations, needed to realize all kernels listed in Table 4.1.

6. Proposed generalized multiply-update operation. Let us now summarize the main points made thus far. First, future parallel computers will involve hundreds of millions (if not billions) of FMA units in a single (super)computing system. Such systems are likely to remain within the same (inflation adjusted) price range as today's largest supercomputers. As a result, price per FMA unit will be further substantially reduced (likely to reach less than 1 US cent per GFlop). Second, unless a great progress is going to be made in the area of quantum computing, the limitations imposed by physics will not go away, keeping the design of computer hardware in a "physics box." Furthermore, it is unclear if quantum computing will be applicable to all classes of computational problems. As a result, *simplicity and uniformity of data movement* has to become the guiding principle of design of both hardware and software for solving computationally intensive problems. This being the case, we predict a diminishing role of divide-and-conquer methods. Here, we mean approaches focused on replacing multiplications with additions/subtractions, and/or reduction of the total number of arithmetical operations, while introducing complex data movement and need for extra memory. Both of them are precisely the problems that stand in the way to development of exa/zetta-flop computers. Third, sometimes without realizing this, scientists solving large number of computational problems, have been working within algebraic semirings. Algebra of semirings involves not only standard linear algebra, but also a large class of other APPs.

Solutions to these problems involve execution of a large number of matrix "multiply-and-add" operations rooted in generalized scalar FMA operations. In this context, we have illustrated the positive effects of development of FMA processing units, and discussed potential benefits of "upgrading" such units to become *generalized FMA units*, capable of dealing with semiring-defined operations listed in Table 4.1. Finally, we have stressed that simplicity of data movement /access concerns also simplicity of code writing. In other words, it seems that the world of scientific computing has reached the era of Ptolomeic epicycles, investigating methods, which are effective in squeezing performance of existing supercomputers, but are going to be ineffective in a long run.

Based on these considerations, we can define a generic form of the matrix "multiply-and-update" (MMU) operation

$$\mathtt{C} \leftarrow \mathtt{GMMU}[\otimes,\oplus](\mathtt{A},\mathtt{B},\mathtt{C}) : \mathtt{C} \leftarrow \mathtt{C} \oplus \mathtt{A}^{\mathtt{T/N}} \otimes \mathtt{B}^{\mathtt{T/N}}, \tag{6.1}$$

where the $[\otimes,\oplus]$ operations originate from different matrix semirings, while `T/N` denotes the fact the either (or both) matrix(ces) A and/or B can be used in a transposed form. Note that, like in the scalar FMA operations, the generalized matrix *addition* or *multiplication*, can be implemented by making matrix A (or B) $= \bar{I}$ for addition, or matrix $C = \bar{0}$ for multiplication (where, the appropriate $\bar{0}$ and $\bar{I}$ matrices have been defined in Section 4).

Finally, observe that the proposed approach leads to new ways of development (design and implementation) of efficient codes solving variety of APPs. Upon reflection, it should also become clear that this approach can be seen as a generalization of the level 3 BLAS. In Section 2 we have discussed the development of the BLAS standard, in the context of development of computer hardware, and growing understanding of the nature of computational linear algebra. We have also mentioned that the BLAS standard did not move smoothly from Fortran to the object oriented languages. Let us therefore, summarize the state-of-the-art in the area of object oriented BLAS and object oriented libraries for computational linear algebra.

7. State-of-the-art in object oriented BLAS. While our work extends and generalizes matrix multiplication, taking into account changes in computing that took place within last 30 years, its current realization should be object oriented. Therefore, let us start from a summary of selected object oriented realizations of numerical linear algebra in general, and BLAS in particular: MTL [11], uBLAS [4], TNT [8], Armadillo [3] and Eigen [7]. Other object oriented projects that could be considered pertinent to the material presented here, have been summarized in [54].

The uBLAS project [4] was focused on design of a C++ template class library for BLAS level 1, 2 and 3; for dense, packed, and sparse matrices. The primary goal of uBLAS was to provide usable software for the scientific computing community. However, its additional goal was to experimentally evaluate if the abstraction penalty, resulting from object orientation (use of matrix and vector classes), is acceptable. According to the information available at [4], the design of the uBLAS was guided by results originating from the following projects: (i) Blitz++ [1] by Todd Veldhuizen, (ii) POOMA [2] by Scott Haney et al., and MTL [11] by Jeremy Siek et al. Data found on the Web indicates that the project was completed around 2002 and later its results have been included in the BOOST [6] library of object oriented mathematical software. Comparing Web pages of the 2010 and 2012 releases of BOOST it is clear that the uBLAS is not developed further, but only maintained by the team of the BOOST project.

Year 2004, marks the end of the life cycle of the Template Numerical Toolkit (TNT) project from the NIST. The TNT is a collection of interfaces for sparse and dense matrices. In addition, a C++ reference implementations of these objects are provided. The library, while not updated since 2004, can still be downloaded from the project Web site and experimented with (the site is signed by Roldan Pozo, the last update took place in March 2004, and the project status is described as: active maintenance).

Interestingly, one of the projects that was taken into consideration while developing the uBLAS, the Matrix Template Library (MTL), outlived the explicitly traceable activity of the uBLAS project. Specifically, in 2005 the MTL project moved from US to Germany and changed the lead personnel. Furthermore, when designing the MTL 4, only the main ideas from the MTL 2 were used, but the code has been written from scratch. The MTL 4 team continued publishing research papers until, approximately, 2009. Since then the project was turned into a commercial endeavor through the SimuNova company, which provides open source, supercomputing, and GPU versions of the MTL 4.

Finally, there are two projects that are vigorously pursued today. These are the Armadillo (with the last release on February 20, 2013) and the Eigen (with the last release on November 5, 2012). Both of them provide support for an extensive set of matrix operations as well as other computational kernels (Eigen, in particular). While both libraries are open source, the Armadillo provides direct support for vendor optimized matrix libraries: MKL and ACML. The Eigen, on the other hand, has been optimized for vector processors (and specially optimized for small matrices), and accommodates large body of supplementary community implemented codes.

8. Initial object oriented model of generalized matrix multiplication. Following the ideas underlying the above summarized projects, as well as the main points discussed above, in Figure 8.1 we depict our proposed initial object oriented model for the generalized matrix multiplication. The proposed model is language independent, so that it will have to be appropriately adjusted if it is to be actually implemented, for instance, in C++ or Java. Here, and in the next section, we follow the lead of the creators of the BLAS standard, who in [44, 28, 27] have not only discussed the general ideas concerning BLAS, but also introduced some ideas concerning its implementation.

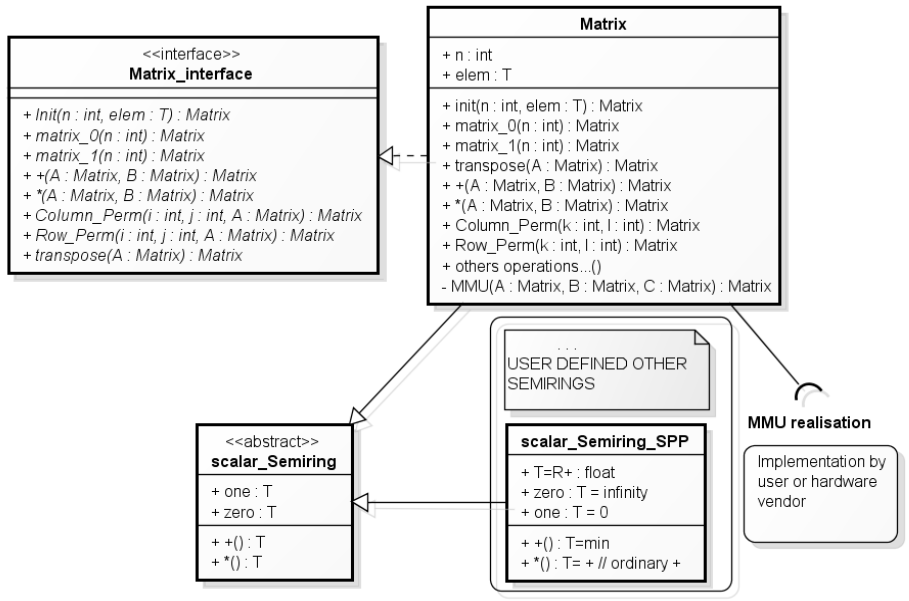

FIG. 8.1. *General schema of the proposed object oriented model of MMU matrix multiplication*

The main class of the proposed model is the *Matrix* class. This class is based on the user defined semirings, which are represented in the *scalar_Semiring* class. In Figure 8.1, we represent the fact that users can define different semirings. Specifically, defining a semiring involves defining the abstract (generalized) operations *addition* and *multiplication* (and *closure* for closed semirings), as well as special elements $\overline{0}$ and $\overline{1}$ (see, Section 4 and Table 4.1). In the case of the *Matrix* class, we can see that the matrix is defined by two generic parameters: matrix size N and type of its elements.

The *scalar_Semiring* class is the basis for the MMU operation (it defines scalar operations within the generalized MMU). Our assumption is that implementation of this operation will be vendor-supported (in a similar way that the BLAS operations have been implemented by the Cray or the IBM). For instance, if specific scalar FMA operations, corresponding to the user defined semiring, are available in the hardware, they will be utilized in the implementation of the MMU operation. Furthermore, if parallel implementation of the MMU is supported on a given computer hardware (e.g. on a GPU), it will be used here.

9. Sample realization. Let us now outline the main features of the proposed realization of the above described approach. Here, let us observe that one of our important goals is to simplify code development (in a

way that the BLAS standard simplified it 30+ years ago). This means, that we want the code to be written by the programmer to be as simple as possible, with most details of the implementation hidden from her. In what follows, we distinguish between the *user*, who will use the proposed model to write codes to solve her problem and the *implementer* who will develop complete model of the generalized MMU. With this in mind, let us start from defining the interfaces. The first one of them is the interface `Matrix_interface`. This interface defines operations that are made available to the user to write his codes.

```
/* T - type of matrix element */

interface Matrix_interface {
  Init(n);/initialisation of square matrix nxn
  Matrix matrix0(n) {/*generalized zero matrix*/}
  Matrix matrix11(n) {/*generalized identity matrix*/}
  Matrix operator + /*generalized matrix addition A+B*/
  Matrix operator * /*generalized matrix product A*B*/
  Matrix transpose(A) {/*transposition of matrix A*/}
  /*generalized permutation of column i and j in matrix A*/
  Matrix Column_Permutation (A,i,j)
  /*generalized permutation of row i and j in matrix A*/
  Matrix Row_Permutation (A,i,j)
...
}
```

As we can see, the implementer is provided with ability to create matrix objects, as well and zero and identity matrices (for a given semiring). Furthermore, we define generalized multiply and add operators, as well as two permutation matrices. This interface can be extended to include other matrices / operations needed by the user.

The second interface is the `scalar_Semiring_interface`. This is where the scalar semiring is specified.

```
interface scalar_Semiring_interface{
  /*Operations:*/
    +,*
  ...
}

abstract class scalar_Semiring {
public:
  //T -- type of element;
  zero,one:T;
    +: c=a+b;
    *: c=a*b;
}
```

Here, the generalized scalar operations $+$ and $*$, as well as scalar elements $\bar{0}$ and $\bar{1}$ are specified, in the abstract class `scalar_Semiring`. Specification of this class has to be provided by the user to define the semiring that she would like to work with in her code.

With these two interfaces in place we can now define the core class `Matrix`. This class is *not* made visible to the user (it is *internal* to the proposed realization of the generalized MMU). It inherits the `scalar_Semiring` interface and implements the interface `Matrix_interface`.

```
class Matrix inherit scalar_Semiring implement Matrix_interface {

  T: type of element;/*double, single,...*/
  n:int;
  // Methods
  Init(n);/initialisation of square matrix nxn
  Matrix matrix_0(n) {/*0 matrix*/}
```

```
8   Matrix matrix_1(n) {/*identity matrix*/}
9   Matrix matrix_1P(i,j,n){
10  /*identity matrix with interchanged columns i and j*/
11  }
12  Matrix transpose(A:Matrix){/*MMU-based transposition of A*/}
13  Matrix operator + {A,B:Matrix}
14    {return MMU(A,matrix_1(n),B,a,b)}
15  Matrix operator * {A,B:Matrix}
16    {return MMU(A,B,matrix_0,a,b)}
17  Matrix Column_Permutation (A,i,j){
18    P=matrix_1P(i,j,n);
19    O=matrix_0(n);
20     return MMU(P,A,O)}
21  Matrix Row_Permutation (A,i,j){
22    P=matrix_1P(i,j,n);
23    O=matrix_0(n);
24     return MMU(A,P,O)}
25  ...
26  private MMU(A,B,C:Matrix(n)){
27    return "vendor/implementer specific realization of
28        MMU = C + A*B where
29      + and * are from class scalar_Semiring"}
30 ...
31 }
```

The most important part of this class is the private function MMU. This is the actual realization of the MMU operation (see, equation 6.1). It is implementer/vendor specific. In other words, user can perform matrix operations: $A \oplus B$, $A \otimes B$, or $C \oplus A \otimes B$, written in the code as $A + B$, $A * B$, or $C + A * B$ without any knowledge that they are actually realized through invocation of the MMU function. Furthermore, the MMU function can be implementer specified or hardware vendor provided, and be based on any of the existing matrix multiplication algorithms (see, section 3).

As far as the matrices in transposed form are concerned, we have to distinguish two senses in which the transpose can materialize. First, transpose may appear as an operation that has to be actually performed on a matrix. In this case, we will apply the $tanspose(A)$ operation (available within the *Matrix* class; see, also [64]). However, transpose may also appear within the context of the MMU operation. Here, as shown in [62], as long as the transpose concerns only one of the two matrices (A or B), it is possible to complete the MMU operation without actually transposing the matrix. Therefore we assume here, that the compiler confronted with code involving $C \leftarrow C + transpose(A) * B$ or $C \leftarrow C + A * transpose(B)$, will call the appropriate implementation of the MMU. In the case, when the user asks for $C \leftarrow C + transpose(A) * transpose(B)$, the actual transpose will be performed on one of the two matrices (e.g. $transpose(A)$) and then the appropriate MMU implementation will be called to complete the operation (e.g. $C \leftarrow C + A * transpose(B)$).

Observe also, that matrix column permutation and matrix row permutation have been defined as operations `Column_Permutation` and `Row_Permutation`, which are implemented through a call to the MMU function with appropriate matrices (see, also [69]).

In the next snippet we show the class `scalar_Semiring`, rewritten for the Shortest Path Problem (point 2, in Figure8.1). After defining this class the user can simply apply the MMU operation within his implementation of the solver. Obviously, this operation can be applied to the whole matrix or to appropriate blocks (in a blocked solver).

```
1
2 /*T = R+ PLUS infinity, general add = min,
3 general product = +, ZERO = infinity, ONE = 0; */
4
5 class scalar_Semiring {
6   zero="infinity";
```

```
 7   one=0;
 8   scalar operator +(a,b:T){return min(a,b)};
 9   scalar operator *(a,b:T){return a+b};
10 }
```

Observe that after this definition, operation defined in the code as $C + A * B$ (with or without the transpose) will be performed within the needed semiring. Obviously, similar definitions can be easily instantiated on the basis of the remaining semirings listed in Table 4.1, as well as semirings materializing in other AAPs.

10. Concluding remarks. The aim of this paper was to summarize and extend recent research results concerning role of dense matrix multiplication in scientific computing. First, it was argued that with the decreasing price of arithmetical operations and abundance of memory, and the increasing cost of data access / move, the old approaches that focused on reducing number of arithmetical operations and memory use need to be reconsidered. Second, it was shown how the Algebraic Path Problem unifies different problems through the theory of algebraic semirings. This allowed to define the generalized matrix multiply and update operation. Finally, a language independent object oriented model of this operation was presented. In the near future we plan to combine the proposed approach with the, recently introduced, mesh-of-tori based systems (see, [63]) to show how it can be effectively applied to a larger class of matrix "manipulations" implemented through matrix multiplications.

Acknowledgment. Work of Marcin Paprzycki was completed while visiting the University of Aizu.

REFERENCES

[1] http://blitz.sourceforge.net/.
[2] http://acts.nersc.gov/pooma/.
[3] Armadillo c++ linear algebra library. http://arma.sourceforge.net/.
[4] Basic linear algebra. http://www.boost.org/doc/libs/1_35_0/libs/numeric/ublas/doc/index.htm.
[5] Blacs.
[6] Boost c++ liberaries. http://www.boost.org/.
[7] Eigen wiki. http://eigen.tuxfamily.org/index.php?title=Main_Page.
[8] Template numerical toolkit. http://math.nist.gov/tnt/.
[9] Top500 list – june 1993. http://www.top500.org/list/1993/06/?page=2.
[10] Motorola x8 mobile computing system. http://www.motorola.com/us/X8-Mobile-Computing-System/x8-mobile-computing-system.html, 2013.
[11] Overview of mtl4. http://www.simunova.com/en/node/24, 01 2013.
[12] S. Kamal Abdali and B. David Saunders. Transitive closure and related semiring properties via eliminants. *Theor. Comput. Sci.*, 40:257–274, November 1985.
[13] R. Agarwal, F. Gustavson, and M. Zubair. A high performance matrix multiplication algorithm on a distributed-memory parallel computer, using overlapped communication. *IBM J. of Res. and Develop.*, 38(6):673–681, 1994.
[14] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide (third ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.
[15] David H. Bailey, King Lee, and Horst D. Simon. Using Strassen's algorithm to accelerate the solution of linear systems. *J. Supercomputing*, 4:357–371, 1991.
[16] V. Batagelj. Semirings for social network analysis. *Journal of Mathematical Sociology*, 19(1):53–68, 1994.
[17] K. Boland and A. Dollas. Predicting and precluding problems with memory latency. *IEEE Micro*, 14(4):59–67, 1994.
[18] J.D. Bruguera and T. Lang. Floating-point fused multiply-add: reduced latency for floating-point addition. *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pages 42–51, June 2005.
[19] L.E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm*. PhD thesis, Montana State University, 1969.
[20] Siddhartha Chatterjee, Leonardo R. Bachega, Peter Bergner, Kenneth A. Dockser, John A. Gunnels, Manish Gupta, Fred G. Gustavson, Christopher A. Lapkowski, Gary K. Liu, Mark P. Mendell, Rohini D. Nair, Charles D. Wait, T. J. Christopher Ward, and Peng Wu. Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L. *IBM Journal of Research and Development*, 49(2-3):377–392, 2005.
[21] Jaeyoung Choi, James Demmel, Inderjit S. Dhillon, Jack Dongarra, Susan Ostrouchov, Antoine Petitet, Ken Stanley, David W. Walker, and R. Clinton Whaley. Scalapack: A portable linear algebra library for distributed memory computers - design issues and performance. In *PARA*, pages 95–106, 1995.
[22] Jaeyoung Choi, Jack Dongarra, Susan Ostrouchov, Antoine Petitet, David W. Walker, and R. Clinton Whaley. A proposal for a set of parallel basic linear algebra subprograms. In *PARA*, pages 107–114, 1995.
[23] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[24] Marius Cornea, John Harrison, and Ping Tak Peter Tang. Intel Itanium Floating-point architecture. In *WCAE'03: Proceedings of the 2003 workshop on Computer Architecture Education*, page 3, New York, NY, USA, 2003. ACM.
[25] Cliff Cyphers and Marcin Paprzycki. Multiplying matrices on the cray-practical considerations. *CHPC Newsletter*, 6(6):77–82, June 1991.
[26] Paolo D'Alberto and Alexandru Nicolau. Using recursion to boost ATLAS's performance. In *ISHPC*, pages 142–151, 2005.
[27] J. J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1990.
[28] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Software*, 14:1–17, 1988.
[29] G. Fox, S. Otto, and A. Hey. Matrix algorithms on a hypercube I: Matrix multiplication. *Parallel Computing*, 4:17–31, 1987.
[30] Maria Ganzha, Stanislav Sedukhin, and Marcin Paprzycki. Object oriented model of generalized matrix multipication. In *FedCSIS*, pages 439–442, 2011.
[31] Kazushige Goto and Robert van de Geijn. High Performance Implementation of the Level-3 BLAS. *ACM Transactions on Mathematical Software*, 35(1).
[32] Brian Grayson and Robert Van De Geijn. A high performance parallel Strassen implementation. *Parallel Processing Letters*, 6(1):3–12, mar 1996.
[33] NVIDIA Group. Tesla K20X GPU Accelerator. Technical report, NVIDIA, November 2012.
[34] John L. Gustafson. Algorithm leadership. *HPCwire*, Tabor Communications, April 06, 2007.
[35] Fred G. Gustavson, José E. Moreira, and Robert F. Enenkel. The fused multiply-add instruction leads to algorithms for extended-precision floating point: applications to java and high-performance computing. In *CASCON '99: Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, page 4. IBM Press, 1999.
[36] Nicholas J. Higham. Exploiting fast matrix multiplication within the level 3 BLAS. *ACM Trans. Math. Softw.*, 16(4):352–368, December 1990.
[37] S. Hunold, T. Rauber, and G. Rünger. Combining building blocks for parallel multi-level matrix multiplication. *Parallel Comput.*, 34(6–8):411–426, 2008.
[38] IBM. Essl library, 2014.
[39] Cray Research Inc. Unicos math and scientific library reference manual sr-2081, 1990.
[40] A. Kamel, M. Kindelan, and P. Sguazzero. Seismic computations on the IBM 3090 vector multiprocessor. *IBM Systems journal*, 27(4):510–527, 1988.
[41] Vladik Kreinovich. Itanium's new basic operation of fused multiply-add: theoretical explanation and theoretical challenge. *SIGACT News*, 32(1):115–117, 2001.
[42] S.Y. Kung. *VLSI Array Processors*. Prentice Hall, 1988.
[43] Jakub Kurzak, Piotr Luszczek, Mark Gates, Ichitaro Yamazaki, and Jack Dongarra. Virtual systolic array for qr decomposition. Technical report, University of Tennessee, 2012.
[44] C. L. Lawson, R. J. Hanson, R. J. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Software*, 5:308–323, 1979.
[45] Daniel J. Lehmann. Algebraic structures for transitive closure. *Theor. Comput. Sci.*, 4(1):59–76, 1977.
[46] J. Li, A. Skjellum, and R. D. Falgout. A poly-algorithm for parallel dense matrix multiplication on two-dimensional process grid topologies. *Concurrency: Practice and Experience*, 9(5):345–389, 1997.
[47] Jin Li, Anthony Skjellum, and Robert D. Falgout. A poly-algorithm for parallel dense matrix multiplication on two-dimensional process grid topologies. *Concurrency - Practice and Experience*, 9(5):345–389, 1997.
[48] O. Lindtjorn, R. Clapp, O. Pell, Haohuan Fu, M. Flynn, and Haohuan Fu. Beyond traditional microprocessors for geoscience high-performance computing applications. *Micro, IEEE*, 31(2):41–49, March-April 2011.
[49] Benjamin Lipshitz, Grey Ballard, James Demmel, and Oded Schwartz. Communication-avoiding parallel Strassen: implementation and performance. In *SC*, page 101, 2012.
[50] Michele Martone, Salvatore Filippone, Pawel Gepner, Marcin Paprzycki, and Salvatore Tucci. Use of hybrid recursive csr/coo data structures in sparse matrices-vector multiplication. In *IMCSIT*, pages 327–335, 2010.
[51] Kazuya Matsumoto and Stanislav G. Sedukhin. A solution of the all-pairs shortest paths problem on the cell broadband engine processor. *IEICE Transactions*, 92-D(6):1225–1231, 2009.
[52] R. K. Montoye, E. Hokenek, and S. L. Runyon. Design of the IBM RISC system/6000 floating-point execution unit. *IBM J. Res. Dev.*, 34(1):59–70, 1990.
[53] Robert K. Montoye, Erdem Hokenek, and Stephen L. Runyon. Design of the IBM RISC System/6000 Floating-Point Execution Unit. *IBM Journal of Research and Development*, 34(1):59–70, 1990.
[54] Claire Mouton. A study of the existing linear algebra libraries that you can use from c++ (une étude des bibliothèques d'algèbre linéaire utilisables en c++). *CoRR*, abs/1103.3020, 2011.
[55] S.M. Mueller, C. Jacobi, H.-J. Oh, K.D. Tran, S.R. Cottier, B.W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S.H. Dhong. The vector floating-point unit in a synergistic processor element of a cell processor. *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pages 59–67, June 2005.
[56] Frank P. O'Connell and Steven W. White. Power3: The next generation of PowerPC processors. *IBM Journal of Research and Development*, 44(6):873–884, 2000.
[57] V. Pan. Strassen's algorithm is not optimal: Trililnear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *FOCS*, pages 166–176, 1978.
[58] Marcin Paprzycki. Comparison of Gaussian elimination algorithms on a Cray Y-MP. *Linear Algebra and Its Applications*, 172:57–69, 1992.
[59] Marcin Paprzycki. Parallel matrix multiplication-can we learn anything new? *CHPC Newsletter*, 7(4):55–59, February 1992.

[60] Marcin Paprzycki. Parallel Gaussian elimination algorithms on a CRAY Y-MP. *Informatica*, 19(2):235–240, 1995.
[61] Marcin Paprzycki and Cliff Cyphers. Using strassen's matrix multiplication in high performance solution of linear systems. *Journal of Computers in Mathematics Applications*, 31(4/5):55–61, 1996.
[62] Abhijeet A. Ravankar. A new "mesh-of-tori" interconnection network and matrix based algorithms. Master's thesis, University of Aizu, September 2011.
[63] Abhijeet A. Ravankar and Stanislav G. Sedukhin. Mesh-of-tori: A novel interconnection network for frontal plane cellular processors. In *ICNC*, pages 281–284, 2010.
[64] Abhijeet A. Ravankar and Stanislav G. Sedukhin. An O(n) time-complexity matrix transpose on torus array processor. In *ICNC*, pages 242–247, 2011.
[65] Sara Robinson. Towards an optimal algorithm for matrix multiplication. *SIAM News*, 38(9), 2005.
[66] Richard M. Russell. The Cray-1 computer system. *Commun. ACM*, 21:63–72, January 1978.
[67] S. G. Sedukhin, T. Miyazaki, and K. Kuroda. Orbital systolic algorithms and array processors for solution of the algebraic path problem. *IEICE Transactions on Information and Systems*, E93.D(3):534–541, 2010.
[68] Stanislav G. Sedukhin and Toshiaki Miyazaki. Rapid*closure: Algebraic extensions of a scalar multiply-add operation. In *CATA*, pages 19–24, 2010.
[69] Stanislav G. Sedukhin and Marcin Paprzycki. Generalizing matrix multiplication for efficient computations on modern computers. In *PPAM (1)*, pages 225–234, 2011.
[70] Stanislav G. Sedukhin, Ahmed S. Zekri, and Toshiaki Myiazaki. Orbital algorithms and unified array processor for computing 2D separable transforms. *Parallel Processing Workshops, International Conference on*, 0:127–134, 2010.
[71] Edgar Solomonik, Bhinav Bhatele, and James Demmel. Improving communication performance in dense linear algebra via topology aware collectives. In *Proceedings of the ACM/IEEE Supercomputing Conference*. ACM, November 2011.
[72] Fengguang Song, Shirley Moore, and Jack Dongarra. Experiments with Strassen's algorithm: from sequential to parallel. In *International Conference on Parallel and Distributed Computing and Systems (PDCS06)*. ACTA Press, 2006.
[73] V. Strassen. Gaussian Elimination is Not Optimal. *Numerische Mathematik*, 14(3):354–356, 1969.
[74] Akihito Takahashi and Stanislav Sedukhin. Parallel blocked algorithm for solving the algebraic path problem on a matrix processor. In *HPCC*, pages 786–795, 2005.
[75] Robert van de Geijn and Jerrell Watts. SUMMA: scalable universal matrix multiplication algorithm. Technical Report TR-95-13, The University of Texas, apr 1995.
[76] R. Clinton Whaley, Antoine Petitet, and Jack Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1-2):3–35, 2001.
[77] Wikipedia. Flops. `http://en.wikipedia.org/wiki/FLOPS`.
[78] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Parallel Comput.*, 35:178–194, March 2009.
[79] Syoudai Yokoyama. Porting matrix inversion and 2D DFT algorithms to Cell/B.E. Master's thesis, The University of Aizu, Japan, 2011.
[80] Field G. Van Zee and Robert A. van de Geijn. Blis: A modern alternative to the BLAS. Technical report, University of Texas, 2012.

Edited by: Dana Petcu
Received: May 1, 2014
Accepted: Jun 24, 2014

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2$_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.