AGM: A DSL for Mobile Cloud Computing Based On Directed Graph

Nikola Tanković* and Tihana Galinac Grbac†

*Superius d.o.o., Pula
†Faculty of Engineering, University of Rijeka

Abstract. This paper summarizes a novel approach for consuming a domain specific language (DSL) by transforming it to a directed graph representation persisted by a graph database. Using such specialized database enables advanced navigation trough the stored model exposing only relevant subsets of meta-data to different involved services and components. We applied this approach in a mobile cloud computing system and used it to model several mobile applications in retail, supply chain management and merchandising domain. These application are distributed in a Software-as-a-Service (SaaS) fashion and used by thousands of customers in Croatia. We report on lessons learned and propose further research on this topic.

Keywords: mobile cloud computing, domain specific language
PACS: Computer languages, 07.05.Bx

INTRODUCTION

Retail, supply chain management (SCM) and merchandising domain is full of data based applications that would significantly benefit from runtime adaptations to end user requirements that are propagated over all integrated business applications. SCM process involves a logistic network from producer organization that distribute its products to retails. This process is negotiated and agreed by service level agreement (SLA) contract for which the further realization is carefully planed and managed during the contract execution. All non-compliances are subject to penalties and are very sensitive for business operations. From the contract a concrete and measurable objectives are used to monitor the SCM process. Whole management process of contract execution is implemented within IT products. Such IT products needed software engineering effort and adaptations for each new customer agreement. Moreover, the software engineers were involved into contract negotiation process and thus had very unpleasant role in the whole operation process. However, the contract agreements are usually similar one to another and may be described by domain specific abstractions derived from real use cases. Such abstractions could be integrated within common software and generated per customer contract basis. The contracts may be modeled by the end users, similarly to idea of programming by example presented in [1], automating the whole process. The whole solution could then migrate to cloud, and be provided to variety of customers, which motivated this research.

In this paper we present a technological solution for Model Driven Development (MDD) of SCM applications. A solution is derived from experience in developing SCM applications for Croatian small-medium enterprises. Because there is a number of issues concerning SCM applications, we focused on presenting the basic technical architecture derived from experienced programmers and personal views.

TECHNICAL APPROACH

As we disseminated in previous research [2], we decided to pursue a interpretative MDD approach [3] meaning that majority of model interpretation is achieved at run-time. For that reason we needed a robust set of functionality for retrieving different aspects of central application model used by many simultaneous user processes. Since we also wanted to enable run-time changes to this central model in order to achieve a End-User development system [4], we needed to ensure isolated and atomic set of updates. This is why a $Neo4j$ graph database [5] came as a natural fit providing ACID properties in manipulating with graph data. Our graph model is based on Object Process Methodology notation and shares OPM meta-model. OPM is a holistic graphical modeling language applicable to a large variety of domains [6]. The main advantage for choosing OPM was its ability to capture dynamic and static aspects of a

system within a single model, unlike UML, where different aspects are modeled within separate diagrams. Graph representation has a one-on-one mapping with a DSL language we designed to feed the model.

IMPLEMENTATION

In this section we will lay out the basics of our approach. First subsection will give a brief formal definition of AGM graph representation, and show the advantages of using a graph background for model representation. Next section will discuss technical architecture behind our mobile cloud solution, and third subsection will give the basic usage scenario.

Definitions

Formally, AGM graph is an ordered quadruple of edges, vertices and their respective mapping functions, noted as $G = (E, V, T_E, T_V)$, with $E = \{x_1, x_2, x_3, \ldots, x_n\}, x_i \in \mathscr{D}$, where $\mathscr{D}$ represents our modeled domain; whereas $V \subseteq X \times X$ represents set of ordered vertices $V = \{(x_i, x_j) | i \neq j, x_i, x_j \in E\}$ represent vertex directed from x_i to x_j. Mapping functions are defined as $T_E : E \to \mathscr{D}_E$, representing type of node from OPM meta-model, and mapping $T_V : V \to \mathscr{D}_V$ representing types of vertices. Currently in AGM, $\mathscr{D}_E = \{Object, Process, Event, Layout, \ldots\}$ and $\mathscr{D}_V = \{Association, Aggregation, Inheritance, Creates, Uses, Modifies, \ldots\}$. For a complete list of nodes and vertex types please see [2] or [6].

Object node types are the primary building blocks for a structural aspects of a system, whereas node types like *Process* and *Event* represent dynamic behavior. Currently, AGM provides full modeling capability for structural aspects of system: necessary schemes for persistent storage and user interfaces definition. Structural model is complete, meaning that interpreters can use them with no additional programming code required to render database schemes or user interface. For dynamic definition of the system AGM requires manual implementation of every *Process* node, meaning that there exists an additional mapping from each *Process* nodes to executable artifact or source code. For every unit of this code there is a well defined interface representing what this module is expecting in terms of data. Figure 1 represents a typical block from one of our applications used to sum up total charges in invoicing process. Observe the special functions *input* and *output* for achieving data flow, where inputs and outputs are modeled within AGM.

```
1 var items = input('Invoice Line');
2 var sum = 0;
3 items.forEach(function(item) {
4    sum += new DecimalNumber(item['Retail Total'], 2);
5 });
6 output('Retail Total', sum);
```

FIGURE 1. A sample of source code for *Process* node.

The biggest advantage from classical DSL languages is that by creating a single holistic graph structure, we can use all the well-known concepts from graph theory like graph traversals, graph matching, and querying for sub-graphs. Our graph model interpreters use such capabilities to efficiently and reliably execute end applications. For example, Algorithm 1 is used to build a subgraph representing structural model of desired object. That way, when interpreter wants to retrieve structural view on some object, it can filter our undesired nodes and vertices. Such a narrowed view can then be used e.g. object serialization to XML, or JSON formats, or to generate necessary data manipulation queries in SQL language for communication with relational database.

Apart from mentioned structural view, we also apply *process view* which traverses the model for requested process and returns a set of involved objects which are handled to interpreters to orchestrate the execution. *Layout view* is used to render user interface. As in previous views, graph traversal is used to determine all objects to render for current unit of display (e.g. widget, screen).

Data: $G = (X, U)$, and starting node x_s where $T_E(x_s) \in \{Object, Attribute\}$
Result: $G' = (X', V')$ where $X' \subseteq X$ and $V' \subseteq V$ representing complete structural model of concept x_s
begin
  $X' \leftarrow \{x_s\}, V' \leftarrow \emptyset$
  for $x \in X'$ **do**
    $S_{successors} \leftarrow \{x_j | x_j \in \Gamma^+(x), T_V(x, x_j) = Inheritance\}$
    $X' \leftarrow X' \cup S_{successors}$
    for $s \in S_{successors}$ **do**
      $V' \leftarrow V' \cup (x, s)$
    end
  end
  for $x \in X'$ **do**
    $S_{successors} \leftarrow \{x_j | x_j \in \Gamma^+(x), T_V(x, x_j) \in \{Association, Aggregation\}\}$
    for $s \in S_{successors}$ **do**
      $V' \leftarrow V' \cup (x, s)$
    end
  end
end

Algorithm 1: Traversal for structure view

Architecture and Usage

The model execution is divided among three interpreters: sever-side interpreter (SSI), client-side process execution interpreter (CSPEI), and client-side user interface interpreter (CSUII) depicted in Figure 2. SSI is responsible for creating virtual web service endpoints based upon definition from AGM model. AGM defines input and output parameters for each service. The background implementation of the service is left for software developer. On the client side, there are two main interpreters: SCPEI used as an orchestrator for conducting business processes like collecting orders, and CSUII which is a helper to render UI elements. To minimize communication between client and server, and for situations when client is disconnected from network, we also implemented a local database containing serialized AGM model.

For access to relational database on the server-side there also exists an *Data Adapter* module for manipulating data in persistent storage according to defined structure of data from AGM. We implemented several Data Adapters for different relational databases like PostgreSQL, Oracle and MSSQL.

A generative tool is used for constructing PostgreSQL database schema from AGM model, so that we can keep relational schema synchronized with AGM models. The main difference from classical model-driven approaches is that there is no possibility for artifacts to accidentally deviate from model due to interpretative nature of whole system which ensures that every step of the process is verified with AGM in run-time.

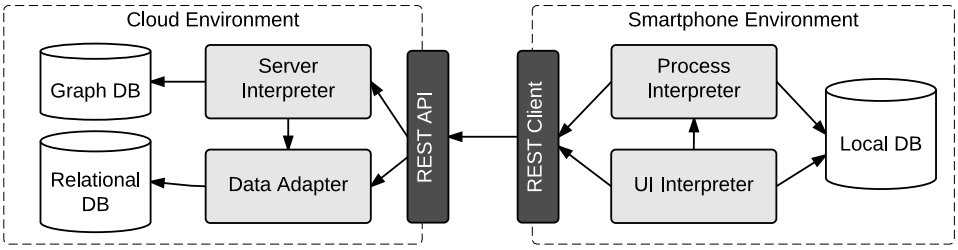

FIGURE 2. High level view on mobile cloud architecture with AGM model

Figure 3 shows a basic usage scenario. The model is loaded by creating a textual representation of domain objects and processes: a simple DSL expressing systems structural and behavioral characteristics which gets automatically translated to a graph structure of AGM. An important part of this process is verification of model. In first step, DSL is only shallowly verified, but when DSL is translated to AGM, we apply a set of graph structure tests like searching for error patterns. Final step requires implementation using a general purpose language (GPL) for each *Process* node.

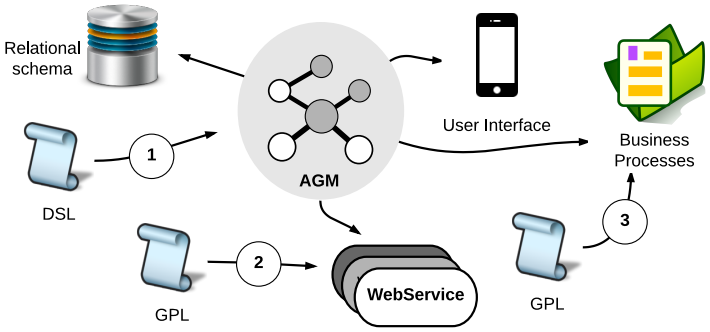

FIGURE 3. Simplified 3 step process for defining AGM application

CONCLUSIONS AND FUTURE ROADMAP

Up to this moment, AGM is used to describe only the functionality of the modeled applications. Little or no concern is given to non-functional aspects. We plan on extending this model to capture the run-time characteristics like QoS and elasticity attributes. This could enable modeling on how the system should be deployed and behave in distributed environment. We plan to integrate TOSCA model within our graph structure and semantically relate it with functional aspects of AGM. That way we could runtime QoS requirements and use such model as a knowledge-base for QoS controllers. We will also make additional efforts to implement a graph analyzing algorithm that could be used to propose optimal deployment strategies [7] based on collecting operational data.

Supply Chain Management process leaves plenty of possibilities for automation, especially concerning software engineering related tasks in capturing requirements. In this paper we propose an technical architecture and its technical description based on Domain Specific Language that is derived from years of experience developing and adapting supply chain application to customer needs. The solution is deployed as cloud application for its customers enabling them to model its application with help of end user development tools provided by our solution. The solution is already used by number of customers in Croatia. In this paper we share some experiences and provide future work.

ACKNOWLEDGMENT

The work presented in this paper is supported by COST action 1304 Autonomous Control for a Reliable Internet of Services (ACROSS) and the research grant 13.09.2.2.16. from the University of Rijeka, Croatia.

REFERENCES

1. S. Gulwani, W. R. Harris, and R. Singh, Spreadsheet data manipulation using examples (2012).
2. N. Tankovic, D. Vukotic, and M. Zagar, Executable Graph Model for building data-centric applications (2011).
3. N. Tankovic, D. Vukotic, and M. Zagar, "Rethinking Model Driven Development: analysis and opportunities," in *Information Technology Interfaces (ITI), Proceedings of the ITI 2012 34th International Conference on*, edited by V. Luzar-Stiffler, I. Jarec, and Z. Bekic, SRCE, 2012, pp. 505–510, ISSN 1334-2762.
4. N. Tankovic, T. Galinac Grbac, and M. Zagar, "Experiences from building a EUD business portal," in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, 2014, pp. 551–556, ISBN 978-953-233-077-9, URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6859629`.
5. J. Miller, *Proceedings of the 2013 Southern Association for ...* pp. 141–147 (2013), URL `http://sais.aisnet.org/2013/MillerJ.pdf`.
6. D. Dori, *Object-Process Methodology: A Holistic Systems Paradigm; with CD-ROM*, vol. 1, Springer Science & Business Media, 2002.
7. N. Tanković, T. Galinac Grbac, H.-l. Truong, and S. Dustdar, "Transforming Vertical Web Applications Into Elastic Cloud Applications," in *International Conference on Cloud Engineering (IC2E 2015)*, IEEE, 2015, pp. 135–144, ISBN 978-1-4799-8218-9, URL `Accepted.http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=7092911`.