ISBN 978-981-11-0008-6
Proceedings of 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016)
Tokyo, Japan, 17-19 June, 2016, pp. 540-544
doi: 10.18178/wcse.2016.06.091

An Agent-Based Framework to Model Integration for Disaster Emergency Management

Qin-Yong Li, Jian-Gong Song, Jiang-Hua Lv [+] and Shi-Long Ma

Beihang University, Beijing, China

Abstract. Model integration is a core issue in the field of disaster emergency management. Models are extremely significant and intelligent resource to improve the ability of decision making in order to alleviate the disaster impacts. Traditionally, practical approach to integrate those models into system is tight-coupling with data sources which lacks of flexibility and re-usability. To cope with this problem, we propose an agent-based loose-coupling framework to model integration and execution for disaster emergency management. We encapsulate models and related, heterogeneous data sources into a standard form using a formal specification. Based on the loose-coupling framework and the standard form, we design a model executing process, which can be divided into five phases and follows the request-reply architecture. Finally, we briefly give a case study based on a real disaster situation: Lu Shan County earthquake in China, to demonstrate the effectiveness and efficiency regarding the proposed framework.

Keywords: model management; decision support systems; disaster emergency management.

1. Introduction

Recently, disasters, such as earthquakes, floods and so on, cause heavy losses on human's life and property. To enhance the ability of emergency response after the disaster occurring is a significant requirement for disaster emergency management. Analysis models [1] to assess disaster situations are extremely essential for improving the ability of disaster emergency response.

Traditional developing methods (tight-coupled method) combine models, data sources and system into an executable program, and then executing the program and getting the solution finally. These methods have several disadvantages as follows: (1) It will hamper system developing schedule if models, data sources and system are tight-coupled; (2) It is hard to modify or to edit model when system is running if models, data sources and system are tight-coupled; (3) It is hard to change or to update data sources if model, data sources and system are tight-coupled.

Researchers and developers have given lots of solutions on model integration in order to develop reusable and flexible systems. Related to model representation and integration, many researchers give more attention on it. A popular way to cope with the problem is to provide a modelling system with some sorts of modelling languages [2]-[4], such as GBMB/SM[5], SMML[6], which is a modelling language based on Structured Modelling [7], etc. Thus, developers use the modelling language to represent the model in order to make models have unified representation. For example, paper [8] proposes a model representation scheme, termed SA-SMML, which is a XML-based language that extends SMML in order to provide semantic-support to model representation.

Unfortunately, pervious studies of model representation/integration have not dealt with some characteristics on model-data independence issue. Pervious proposed work in part lacks of some characteristics such as: (1) analysis models, data sources, and application/system independence; (2) a flexible mechanism with detailed steps to create a model instance with heterogamous data sources.

[+] Corresponding author. Tel.: +86-010-82317643; fax: +86-010- 82317643.
E-mail address: jhlv@nlsde.buaa.edu.cn

To cope with these problems, we propose an agent-based framework to enhance the flexibility and re-usability of model integration for disaster emergency management. In addition, we design a model executing process based on a request-rely architecture.

The paper has been organised in the following way. Section 2 deals with a description of proposed framework and the formal specification to models and data sources. Besides, based on the formal specification, we also describe the process of model executing. In Section 3, we briefly give a case study using this framework in an actual disaster situation: Lu Shan County earthquake. Conclusions and future work will conduct in Section 4.

2. Architecture

2.1. Overview

The proposed framework can be dived into four layers: application layer, proxy layer, middleware layer and resource layer, respectively. Proxy layer and middleware layer constitutes the Agent Module, which is the core component in proposed framework and will discuss in Section 2.3. The architecture of proposed framework is shown in Fig. 1.

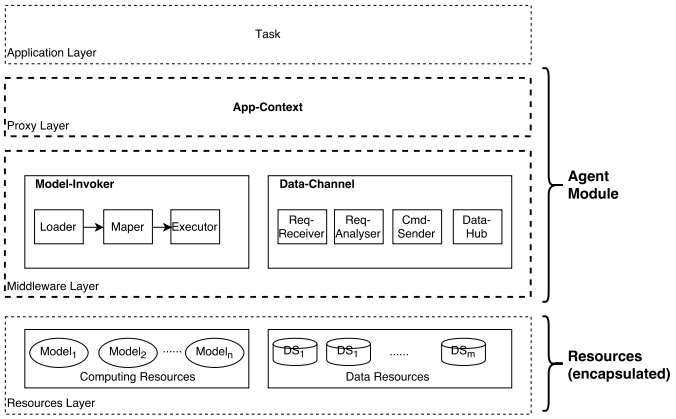

Fig. 1: The architecture of proposed framework.

The application layer is the top layer. This layer's functionality mainly refers to a top object, called Task, which is the analysis target we want to get. In real situation, a task may be a target to assess human fatality, economic losses and so on.

Under the application layer is the proxy layer. This layer's responsibility is to transfer and to interpret messages/data objects between the application layer and the middle ware layer.

The middle ware layer is the core component in our framework. This layer contains two components: Model-Invoker and Data-Channel, to achieve the target which is model executing. Model-Invoker is used to query model, mapping model and data sources (namely, instancing the model), and then executing the model and getting the result. Data-Channel is used to interpret data commands and to interact with data sources in order to get data to support model executing. A detailed description of model executing process can be found in Section 2.4.

The bottom of the framework is the resource layer, which contains analysis resources. Analysis resources refer to two things: Models and data sources, which are two core resources to assess disaster situation. Models and data sources are encapsulated as analysis resources by unified specification. We will briefly give the encapsulating specification in next section.

2.2. Resource encapsulating

2.2.1. Models

A model is defined as a five tuples structure, as follows:

$$\text{Model} = (\text{name}, \langle in_1,\ldots,in_p\rangle, \langle p_1,\ldots,p_q\rangle, \text{PROC}, \langle out_1,\ldots,out_r\rangle);$$

The description for each tuple as follows:

- name: denotes the model's identifier (identifies one and only one model);

- $<in_1,\ldots,in_p>$: denotes a model's input variable sequence; for each $in_i \in <in_1,\ldots,in_p>$, in_i is called a model's input variable;
- $<p_1,\ldots,p_q>$: denotes a model's parameter variable sequence;
- PROC: denotes a model's procedure, which is an executable program or script program; it would be a simple equation, or an explicit function;
- $<out_1,\ldots,out_r>$: denotes a model's output/result variable sequence;

2.2.2. Data sources

A data source comprises a number of data sets. Data sets are organised through a group of data. Data comprises a data-name and a group of data-objects and a collection of data-operations. Generally, a data source is briefly defined as a four tuples structure as follows:

Data-Source = (DS-Name, DS-Type, DS-Addr, Data-Set)

The description for each tuple as follows:

- DS-Name: identifies and only identifies a data-source;
- DS-Type: represents the type of a data-source; In present, we define two types of data source: relational data sources and spatial data sources, which is denoted as DS-Type = {DB, SDB};
- DS-Addr: represents the address of a data source in order to connect to it;
- Data-Set: represents the data which belongs to a data source, Data-Set=$\{Data_1, Data_2,\cdots, Data_n\}$;

2.3. Agent modules

2.3.1. App-context

App-Context comprises a group of data sources, a sequence of business data, and a sequence of initial parameters. App-Context is defined as a three tuples structure:

App-Context = $(\{DS_1,\ldots ,DS_m\},<(a_1, cond_1, db_1),\ldots, (a_n, cond_n, db_n)>, <init_1, \ldots, init_k>)$

The description to each tuple as follows:

- $\{DS_1,\ldots ,DS_m\}$: denotes the data sources;
- $<(a_1, cond_1, db_1),\ldots, (a_n, cond_n, db_n)>$: represents the practical business or parameter data sequence; $a_i = c_i$ represents a single parameter data or a matrix parameter data, while $a_i=b_i$ represents business data; $cond_i$ is a Boolean expression; db_i is the database that contains a_i;
- $<init_1, \ldots, init_k>$: denotes the initial parameter sequence; For example, in an earthquake scenario, it would be <Longitude, Latitude, Magnitude, Depth>;

At the beginning of the model executing, App-Context is empty and determined by Task object when the model being executed. Task component interacts with Model-Invoker and Data-Channel in order to initialize App-Context.

2.3.2. Model-invoker

Model-Invoker is used to query or discovery a model, to check a model whether is compliance to the specification, and to execute a model. Model-Invoker has three sub components:

Model-Invoker = (Loader, Mapper, Executor)

The description to each tuple as follows:

- Loader: to load a model and checking the model whether coined with the encapsulating specification or not;
- Mapper: to establish a link between model's input and/or parameter variables to data sources; in other words, to instance a model;
- Executor: to execute a model and to get the output/result;

2.3.3. Data-channel

Data-Channel is acting as a channel or a mediator between App-Context and Data-Sources.

Data-Channel = (Req-Receiver, Req-Analyser, Cmd-Sender, Data-Hub)

The description to each tuple as follows:

- Req-Receiver: to receive the message sent by App-Context;
- Req-Analyser: to analyse the data request and to translate the requests to data control commands;
- Cmd-Sender: to send the data control commands to Data-Sources;
- Data-Hub: to receive the data which is queried by Data-Source and return the data to App-Context;

2.4. Model executing process

The whole process of model executing mainly divides into five phases, as shown in Fig 2. The proposed process is based on request-reply architecture, which contains request operations, reply operations and inner process operations. We describe each phase as follows:

Phase I. Checking model existence

This phase is designed to check the target-model (want to invoke) whether exists or not. In here, we define Task refers to a model-invoking task. At the beginning, Task sends a query request with target-model's name to App-Context, and then App-Context delivers the request to Model-Invoker. Model-Invoker interacts with Model to check the target-model whether exists or not. Finally, App-Context returns a flag which refers to indicate the target-model existence to Task.

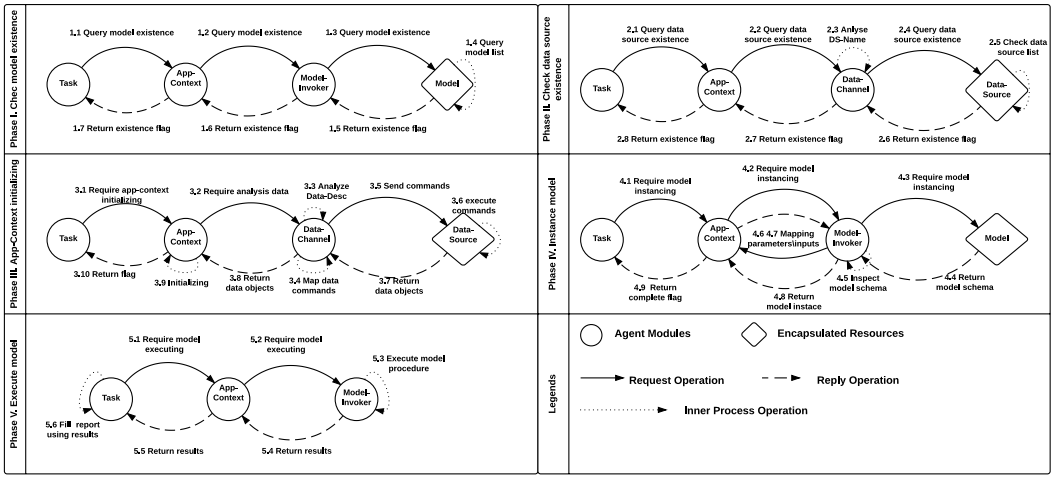

Fig. 2: The process of model executing.

Phase II. Checking data source existence

As similar with Phase I, this phase is designed to check the existence of the related data source for model executing. Task sends request to related components (App-Context and Data-Channel) and gets a reply which indicates the existence of the data source.

Phase III. App-Context initializing

This phase is used to initialize App-Context; namely, filling App-Context using related data in order to support model instancing and executing. App-Context interacts with Data-Channel, and then the Data-Channel translates the data description to practical data commands, such as some sort of SQL scripts for relational databases, or Spatial-SQL scripts for GIS or spatial databases, and sends those commands to Data-Source component. Data-Source executes those commands and returns the result to App-Context. Finally, App-Context uses those data to initialize itself and return a successful flag to Task.

Phase IV. Model instancing

This phase is used to instance a model. Since App-Context has initialized, the model interacts with App-Context in order to instance or to fill its input variables, parameter variables with related data acquired in phase III, according to the model's formal specification in Section 2.2.1. Finally, Task will receive a successful flag if the process of model instancing is complete and no errors occur.

Phase V. Model executing

This phase is used to execute the model which has already instanced by related data. Task sends a request to execute the model and finally gets the result back.

3. Application

The proposed framework in this paper has already applied to China Earthquake Society Service Project (CESSP): a national project is developed by National Earthquake Response Support Service (NERSS). Fig 3 and Fig4 show two examples using this framework to integrate and to execute models in order to estimate the intensity level and the construction damages caused by the Lu Shan County earthquake.

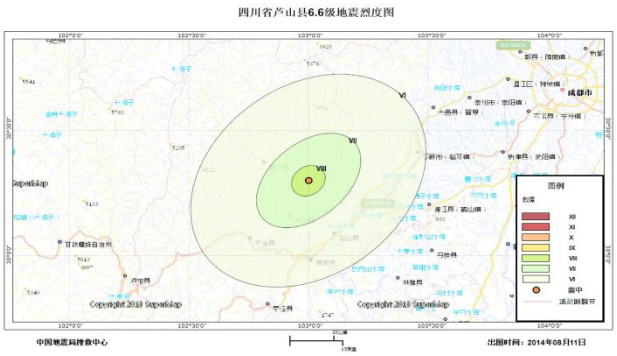

Fig. 3: The estimated intensity level of Lu Shan County Earthquake, China.

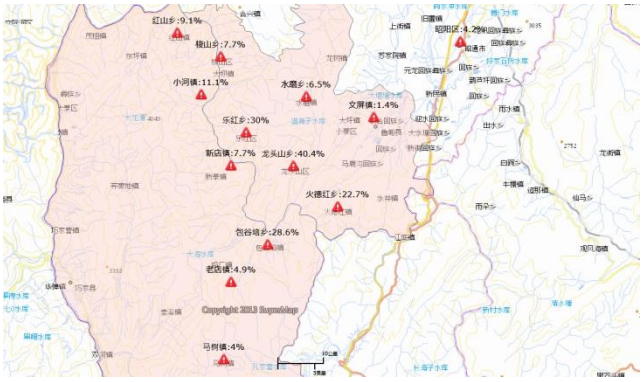

Fig. 4: The estimated construction damage caused by Lu Shan County Earthquake, China.

4. Conclusions and Future Work

Models are essential resources to make effective and efficient decisions. Traditional model developing method in part lacks of flexibility and re-usability. In this paper, we design an agent-based framework to model integration and execution for disaster emergency management. We encapsulate models and data sources into a standard resource form in order to improve system's flexibility and re-usability. Based on our proposed framework and the standard form, we describe the model executing process briefly. Future work is mainly focus on improving the interoperable ability of cross-discipline models in the field of disaster emergency management.

5. Acknowledgements

The authors thank the anonymous reviewers for their insightful and constructive comments. This research work is supported by Social Service Project in National Earthquake Response Support Service "International Rescue and Disposition System against Strong Earthquakes" (NO.SJZX-B11).

6. References

[1] R. Krishnan and K. Chari, "Model management: survey, future research directions and a bibliography," *Interactive Transactions of OR/MS*, 2000.

[2] H. Kim, "An XML-based modeling language for the open interchange of decision models," *Decision Support Systems*, 2001.

[3] T. Bhrammanee and V. Wuwongse, "Towards a Unified Representation Framework for Modelbases and Databases," Decision Support for Global Enterprises, 2007.

[4] T. Bhrammanee and V. Wuwongse, "ODDM: A framework for modelbases," *Decision Support Systems*, vol. 44, no. 3, pp. 689–709, 2008.

[5] K. Chari and T. Sen, "An implementation of a graph-based modeling system for structured modeling (GBMS/SM)," *Decision Support Systems*, 1998.

[6] O. El-Gayar and K. Tandekar, "An XML-based schema definition for model sharing and reuse in a distributed environment," *Decision Support Systems*, vol. 43, no. 3, pp. 791–808, 2007.

[7] A. M. Geoffrion, "An Introduction to Structured Modeling," *Management Science*, vol. 33, no. 5, pp. 547–588, May 1987.

[8] A. Deokar and O. El-Gayar, "On semantic annotation of decision models," *Information Systems and e-Business Management*, 2012.