ABDiSE: Agent-Based Disaster Simulation Environment

Tzu-Liang Hsu and Jane W. S. Liu

***Abstract*—The framework Agent-Based Disaster Simulation Environment (ABDiSE) provides agent-based model elements of common types of natural disasters, including fires, floods and debris flows. Active objects describe how agents move, attach, and interact with each other. The simulation engine provided by the framework enables the objects to be executed for purposes of simulating the causes and dynamics of the disaster modeled by them. ABDiSE is extensible: Agents and external simulators needed to model elements and dynamics of new disaster scenarios and define behaviors and interactions of agents can be added without requiring recompilation.**

***Index Terms*—Agent-based simulation, disaster models, disaster simulation, development environment.**

I. Introduction

In recent years, agent-based modeling (ABM) and simulation have proven to be an effective tool for application domains as diverse as economical trend prediction; complex system design; and fluid, traffic and customer flow analysis [1]-[6]. According to ABM, entities capable of taking actions are modeled as agents. Each agent is defined by a behavior specification and a set of interaction rules. During simulation, each agent assesses its situation, makes decisions and takes actions according to its behavior specification and interaction rules, and through the rules, taking into account the behavior of other agents and states of the environment.

This paper presents an agent-based framework called Agent-Based Disaster Simulation Environment, or ABDiSE for short. The framework contains an extensible library of agents of various types with which one can construct agent-based models of elements that cause natural disasters (including fires, floods and debris flows) and elements of geographical areas (including buildings, trees, and roads) that are affected by disasters. The framework also provides an engine for executing the models and thus simulating the causes and developments of natural disasters for the sake of understanding and predicting their dynamics and impacts. The model elements provided by the framework build on common-sense concepts and notions. By using them, complex processes of how a small mishap (e.g., a cinder touching a dry leave) develops into a major calamity (e.g., a forest fire) can be modeled with the desired fidelity in an intuitive but formal and executable way.

The contributions of this work include the novel agent models. Section III will further elaborate the novel aspects. Our contributions also include the open source framework ABDiSE for disaster simulation. Currently, simulators of natural disasters (e.g., [7]-[14]) are by and large built one at a time, each for a specific type of disaster, locale, disaster situation, and so on. ABDiSE enables simulators for disasters of different types at different places and likely scenarios to be easily constructed from model elements. Specifically, the framework supports common GIS data format and provides easy to use GUI and tools to input information and model elements of the simulation world. Any time after a disaster scenario is setup and simulation progresses, the user can save the current state of the simulation world as a checkpoint and the initial condition for a later experiment. Extensibility is a distinguishing feature of ABDiSE. Agents and simulators needed to model the dynamics of new disaster types and scenarios can be added without modification and recompilation of ABDiSE core controller: When a user wants to add one or more agent types, he/she only needs to write the new agent classes, compile them to DLL (dynamic link library) files, and restart ABDiSE. ABDiSE core controller will dynamically load the DLL functions and thus enable the user to simulate new disaster scenarios modeled by the new type(s) of agents.

The remainder of the paper is organized as follows. Section II presents related work on disaster simulators and agent-based models and simulation. Section III describes ABDiSE agent-based model elements. Section IV describes tools for set up and control simulation experiment. Section V describes the design and architecture of the framework. Section VI describes the ABDiSE simulation engine and the interaction between ABDiSE Core and GUI. Section VII presents as case studies examples illustrating the use of ABDiSE for modeling scenarios of past disasters. Section VIII summarizes the paper.

II. Related Work

There are numerous disaster simulators for specific types of disasters. An example is the advanced fire simulator Fire Dynamics Simulator (FDS) [7]-[9]. The combination of FDS simulation of smoke and heat transport and visualization support from Smokeview program can provide answers to questions on causes of rapid spread of fire and smoke within a building and time required for residents to escape from the building. Similar to FDS, ABDiSE breaks a large-scale simulation into computation of behavior and interactions of elements within small grid cells. The incorporation of solutions of Navier-Stokes equations used in FDS into ABDiSE is possible and is a part of our future work.

Flood simulation can provide data and information on where an inundation may occur, its arrival time, flood wave

Manuscript received August 2015; revised December 1, 2015. This work was supported in part by the Academia Sinica, Sustainability Science Research Project OpenISDM (Open Information Systems for Disaster Management).

T. L. Hsu is with MediaTek, Inc., Hsinchu City 30078, Taiwan (e-mail: lightorz@gmail.com).

J. W. S. Liu is with Institute of Information Science, Academia Sinica, Taiwan (e-mail: janeliu@iis.sinica.edu.tw)..

DOI: 10.7763/IJMO.2015.V5.491

speed and depth for various scenarios critical to effective flood emergency and reclamation management. Numerous flood simulators are now available. Examples include Monticello Dam simulation and generation of inundation map [10] and simulation of Amazon flood pulse based on the 2-D hydrodynamic model LISFLOOD-FP to quantify and predict the exchange between the Amazon main stem and its floodplains [11], [12]. Simulation has also been an approach used to study long term effects of climate change and to create high-resolution, static sea-level rise image of specific area (e.g., Manhattan in New York) for a specific sea level rise value [13], [14]. This is a simple way to estimate flood disaster damage. Similar to these simulators, ABDiSE also supports GIS file format and scenarios creation with Google Earth Map. The default flood simulation is based on the altitude attribute in a way similar to the one used in [13], [14]. Again, the user can incorporate more advanced flood models in ABDiSE by editing the behavior rules of flood agent type and rules of individual flood agent instances and structuring and compiling flood simulation programs into DLLs and use them for update methods of the new flood agents.

Being a framework for the construction and execution of agent-based models of natural disasters, ABDiSE more closely resembles several existing toolkits (e.g., [15]-[18]) for the development of agent-based applications in terms of its approach, goals and design. ABDiSE captures the core ideas of common agent-based models: Agents in the framework have the typical attributes of agents. In addition to interactions such as contention for space and other common interactions, agents in ABDiSE can attach to each other and become joined agents. Joined agents are models of entities such as burning trees, flooded homes, etc. Section 3 will present the definition of attach and its use in modeling dynamics of disasters.

Specifically, the ABDiSE framework was motivated by many existing toolkits [6], [18], including Repast (Recursive Porous Agent Simulation Toolkit) [19], AnyLogic [20], and Natural Disaster Complex Systems Framework [21]. Repast is a widely used open-source, cross-platform, agent-based modeling and simulation toolkit. It supports flexible models of living social agents and models of belief systems, agents, organizations and institutions. Repast allows users to dynamically modify agent properties, agent behavioral equations, and model properties at run time. Moreover, Repast includes libraries of genetic algorithms, neural networks, random number generation and specialized mathematics, as well as supports for social network modeling. ABDiSE has a narrower focus. It specifically aims to make modeling the causes and dynamics of natural disasters conceptually intuitive without loss of rigor and fidelity. ABDiSE also allows users to access and modify agent and environment properties dynamically at run time.

AnyLogic [20] is a multi-method simulation and modeling tool. AnyLogic models can dynamically read and write data to spreadsheets and databases and chart model output during simulation runs. In addition to agent-based modeling, AnyLogic also supports discrete event simulation and system dynamics. ABDiSE does not have these capabilities: It focuses on simulation of natural disasters, offers easy to use tools for agent model extensions, simulation state saver and loader, and run-time editing of agent and environment properties.

Natural Disaster Complex Systems Framework [21] also provides tools for modeling and simulation of natural disasters. In addition to elements for modeling natural disasters, these tools also have elements for modeling and defining organizational structures and the policies that must be taken into account to simulate real-life emergency management activities. Similar to this framework, ABDiSE is also structured to facilitate simulation and observation at different levels of abstraction and details the development of natural disasters. The current versions of ABDiSE do not support the modeling of organizational structures and policies. As pointed out by [22], models of standard operating procedures (SOPs) and human beings can be added to ABDiSE to enable the evaluation of the effectiveness of SOPs in simulated disaster scenarios.

III. Model Elements

As stated earlier, ABDiSE model elements capture common-sense concepts in natural disasters. The goal is for disaster scenario models built from the elements to be conceptually intuitive and easy to understand and use. The model elements are implemented as C# classes.

A. Simulation World, Environment and God

Throughout any simulation experiment setup and carried out within ABDiSE, there is only one simulation world. In the narrow sense, the term world refers to the geographical area specified by the user for the experiment. The world may have multiple regions, each with a specified boundary.

As in real-life, the simulation world has a global environment. Some regions may have local environments that differ from the global environment. Each environment is defined by a set of parameters. At each point in space and time, the values of environment parameters specify attributes and conditions that affect the behaviors of all agents around that point in space and time. The user can choose to provide the parameter values for all or selected grid points and instants or intervals of time. The environment parameters can also be defined by functions of space and time. In a region with a local environment, the behavior of every agent depends on the local environment.

The model for every simulation experiment has one and only one special agent called God. Through this agent, ABDiSE provides the user with capabilities to control over all model elements during each simulation experiment. Conceptually, God can create, and no other agent has this capability. The simulation world, environments and agents running during the experiment are set up (i.e., created and initialized) by this agent. During any experiment, God has the records of all environments and agents in the simulation world. The creation of the instance of God agent is in fact automatically done by ABDiSE GUI: After the GUI pops up, God of the simulation world is ready to carry out user's wishes as specified by the user via the GUI.

In addition to Create, God can Activate, Affect and Control. It is often convenient to create some agents from the start but keep them inactive until some later time instant(s) or

upon the occurrence of some conditions. The update method of an agent is executed during the current simulation step only if the agent is active at the time. Activate method is for this purpose. Using Affect and Control, the user can alter the simulated scenario in ways that cannot be easily or conveniently defined by behavior specifications and interaction rules of individual agents or changes in environments. Examples include a sudden rise in flow volume and temperature to simulate the effect of an upstream dam break and an explosion nearby, respectively.

B. Major Agent Types and Attach Method

Hereafter, we will use the term agent and agent instance interchangeably. In ABDiSE, every agent other than God is of one of two major types: Natural Element Agent Type and Attachable Object Agent Type. Agents of natural element types include cinder, smoke, fluid and so on. Disasters are typically caused by this type of agents. Agents of attachable type are affected by disasters. Examples include tables, buildings, trees and cars. Disasters are due to their interactions with some of natural element agents. Fig. 1 lists some of the essential properties of each agent instance, including flags that indicate the major type of the agent.

AgentType: A string for recording the type (e.g., fire) of the agent
ConfigStrings: Default configuration strings (e.g., common fire classes)
AgentProperties: Detail properties in dictionary<str, str> form (e.g., Name, FireLevel, FireClass)
IsNaturalElementAgent: A Boolean flag
IsAttachableObjectAgent: A Boolean flag
IsJoinedAgent: A Boolean flag

Fig. 1. Essential properties of agents.

Attach is one of agent methods/rules. The Attach rule is applicable to a natural element agent and an attachable object agent (or a joined agent). The result produced by Attach is a joined agent. The natural element agent, or the attachable agent, or both may disappear, and the new joined agent instance thus created inherits their attributes and status. The fact that an agent is a joined agent is indicated by the fact that the value of its IsJoinedAgent flag is true. Depending on the type of the joined agent, the agent may also have its own methods and rules. The user can also customize rules and properties for each joined agent type by customizing the update method of the type.

As an example, we consider a cinder, which is an agent instance of the natural element type, and a table, which is an attachable object. The behavior of the joined agent table-joined-with-cinder changes according to a method of the table agent used to model and simulate the start of a fire from the table and subsequent development of the fire. A forest fire scenario may start from a cinder attaching to a tree and the creation of a new tree-joined-with-cinder agent instance. The behavior of the joined agent is computed by a wild fire simulator. Another example is water attaches to a mud-sand mixture. Mud-sand with a given amount of water attached becomes debris that flows according to a debris flow law or as computed by a debris-flow simulator.

Attach enables us to model events that cause disasters and the development of disasters. As far as we know, no other agent-based model has this rule.

IV. Setup and Control Simulation Experiment

Before describing the structure and implementation of ABiDSE, we use Fig. 2 to help us explain the tools provided by the framework for setting up simulation experiments and control simulation runs. Specifically, Fig. 2 shows a screenshot of the main window of ABDiSE. The user can access from the GUI (Graphical User Interface) tools for managing simulation experiments: These tools enable the user to select and retrieve model elements from the model library and use them to construct the simulation model, set up and control the simulation experiment, and configure the simulation engine.

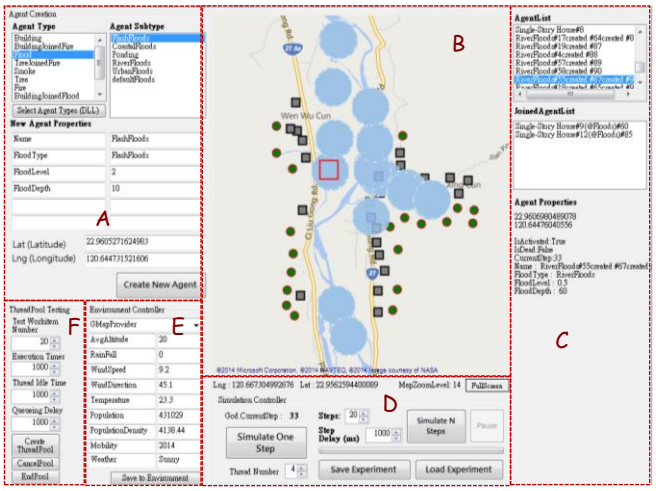

Fig. 2. Main window of ABiDSE GUI.

The most prominently displayed tool is the Map Explorer in area B where a flood affected area is displayed in its entirety as an example. Built on the cross platform and open source .NET control GMAP.NET [23], the tool allows the user to control map attributes, including map provider (Bing, Google, and so on), overlay, marker and zooming. In particular, it provides the user with an easy way to specify locations of agents in a common geographic information system (GIS) format and have them displayed on the map. In this example, small circles and squares in the area represent two types of agents which the user has already selected from the agent library to be included in the simulation model and placed them on the map as indicated. The user can also visualize in Area B the development of the disaster scenario during the simulation run.

Area A is used for selecting agents to be included in the simulation model: To include an agent of a specific type that is provided by the model library, the user only needs to first select an agent type (e.g., flood) and then select a subtype (e.g., flash flood). If none of the available agent types suits the disaster scenario which the user wants to simulate, the user can add a new agent type and/or subtype by clicking the Select Agent Types (DLL) button: This button enables the user to add new agent type(s) or remove some existing type(s) (e.g., add fire type and remove flood type): The user adds a new agent type to the library by programming a customized Dynamic Link Library (DLL) function. The function will be called during simulation to update the behavior of agents of the type. External simulation programs can also be incorporated into the model in a similar way. We will provide

details on the extensibility feature shortly.

Properties of agents are stored in C# dictionary form. When the user selects an agent type for a new agent, the default parameter values of the agent, including the one and only Name property of the agent, is automatically filled in Area A, and an entry for the agent is created in the dictionary. The user can freely assign properties to the new agent. An exception is its Name. Among the important properties of an agent are its coordinates. The user can assign the latitude (lat), longitude (lng) and altitude of a new agent by typing their values in the textboxes in Area A. The values of lat and lng can also be entered by double left clicking its location in Area B. After all required agent properties have been entered, the user then clicks Create New Agent button. Clicking the button creates a new agent according to the agent parameters entered in Area A. If the create operation is successful, the new agent marker will appear in Map Explorer after the entry of the agent in the dictionary is committed. At any point in time, the upper panel in Area C lists all the agent instances included in the simulation model. The lower panel displays the properties of a selected agent (e.g., the one marked by the square in Area B and highlighted in C in Fig. 2.)

Within ABDiSE, simulation is time driven: The engine executes one step at a time, and the update method of each agent is executed once in each step. Using buttons and text boxes in Area D, the user can control the length of the time for each step, as well as the number of time steps to execute during the current simulation. A special model element is Environment. Text boxes in Area E allow the user to set environment properties (e.g., wind speed and direction, rainfall rate, etc.), and thus, provide the user with control over environment during simulation. Finally, text boxes in Area F allow the user to configure and test the simulation engine, in particular, whether the multi-threaded engine executes correctly and performs well.

V. Design and Architecture

ABDiSE is written in C#. It has two versions. Version 1.0 was implemented primarily for the proof-of-concept purpose. Version 2.0 was designed and implemented to provide extensibility: It allows the user to add new agent types and extend the model base without having to revise and recompile the entire framework and simulation engine.

A. Structure

Fig. 3 shows the functional structure of both versions. In addition to libraries of reusable agents, agent-based models, various tools and simulators, the framework provides three major components: They are the graphical user interface (GUI), experiment manager and simulation engine. As described earlier, the user sets up, controls simulation experiments and visualizes simulation results via the GUI, which is implemented using Microsoft Windows Form.

To help the user build an agent-based model of the simulated disaster and sets up the experiment, the Experiment Manager provides select, load and build tools, which are accessible to the user via buttons and text boxes in Area A shown in Fig. 2. The Experiment Manager also allows the user to change the fidelity of the model by adjusting simulation parameters such as spatial and time granularities and choices of simulators. The user can configure the engine by adjusting the number of threads used to run the simulation. The Model Builder loads agent data from agent library to create agent instances. It also loads models, tools, simulation information and facts from libraries and fact database.

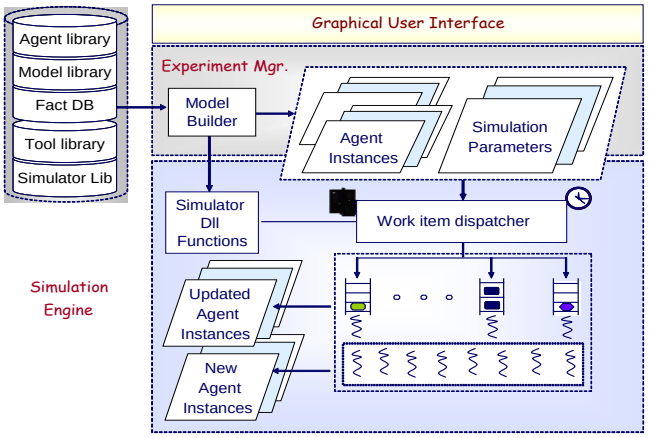

Fig. 3. ABDiSE structure.

The behavior rules of each agent are simulation programs that compute the dynamic behavior of the agent; they are programmed in Update methods of the agent class. During simulation, the Update method of every activated agent instance is executed during each time step. The simulation engine uses a pool of worker threads to do the execution: At the start of each time step, the simulation engine checks the agent list of God for activated agents. It encapsulates the Update method of each activated agent into a work item, and queues the work item into the work item queue in the thread pool. Worker threads in the thread pool dequeue and execute work items. In this way, the states of agent instances and their interactions are updated. In the process, new agents may be created and existing ones may disappear according to their rules, all in time-driven manners.

B. Extensibility

In ABDiSE version 1.0, types of agents are fixed at compile time. Adding new agent types and behaviors requires the user to edit and recompile the code of the simulation environment. This shortcoming was removed in version 2.0 by providing the simulation engine with the ability to load agent types dynamically at initialization time. Specifically, in version 2.0, each agent type is defined by a class. The class inherits the abstract class Agent and implements methods that override abstract methods of Agent. Agent classes are compiled as DLL functions. When ABDiSE 2.0 framework starts, the dynamic loader allows the user to selectively load available agent classes and associated .dll files from the agent library. The user can create new agent types by creating new agent classes for the types and compiling the new classes as DLL functions. Like existing agent classes, the new classes also inherit the abstract class Agent. Their .dll files are also loaded along with the .dll files of existing agent classes. The next section will provide further details on the mechanism.

In addition to enhancing ABDiSE with extensibility, we also restructured the code according to the

Model-View-Controller (MVC) architectural pattern [24] in order to improve maintainability. Fig. 4 shows how the major components of ABDiSE 2.0 fit in the MVC pattern. According to the pattern, classes are divided into three parts: model, view and controller. Model includes agents, joined agents, God, and environments. They directly manage data, logic and rules. Classes in view handle button events and passes user inputs to the controller. Controller manipulates data in the model. It also creates agents by dynamically loading DLL functions, based on rules of the God agent. The core controller uses a pool of worker threads to execute work items, as stated earlier. When the model is updated, the view is refreshed to display updated information.

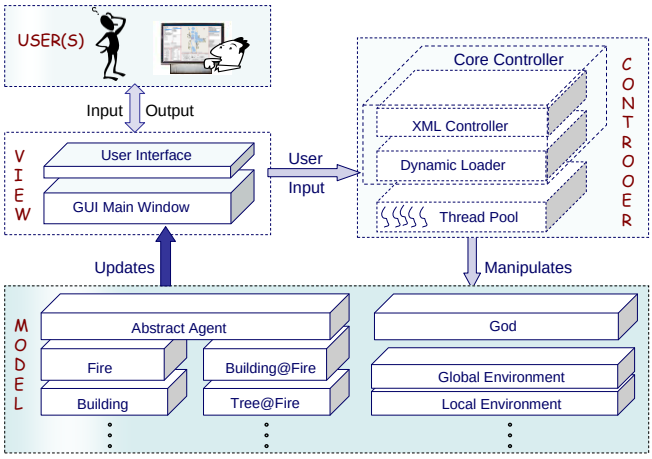

Fig. 4. Major components of ABDiSE.

Due to space limitation, we omit descriptions of classes within ABDiSE. Detailed descriptions of them can be found in our technical report [24].

VI. Core Controller and Simulation Engine

Parts of the code of the constructor of Core Controller and ABDiSE. Controller. Core Controller are listed in Fig. 5. During initialization of ABDiSE 2.0, the constructor of Core Controller creates the one and only one instance of God; initializes the XML file controller; creates instance(s) of Model. Environment with parameters provided by the user via GUI; and launches the dynamic loader; and when the user commands to create agents via the GUI, uses the dynamic loader to load agent instances (i.e., the associated .dll files) from the agent library dynamically.

Suppose that the user has chosen via text boxes in area D of the GUI to run the multi-threaded simulation engine. As a step during start up, the Core Controller creates the thread pool of the simulation engine. The thread pool is a custom thread pool. It includes a user specified number of worker thread(s), a FIFO queue for holding work items waiting to be executed, and methods of controlling worker thread(s).

During simulation, Core Controller interacts with all models elements, including agents, environments, God and the GUI, while the simulation engine uses the worker threads in the thread pool to execute work items. Again, the dynamic behavior of individual agents and joined agents are defined by update methods of their respective types, and update methods of all activated agent instances are wrapped within the work items, invoked as DLL functions and executed by worker threads during each simulation step.

Several parts of the update method need to be guarded by locks. An example is the code for agent attachment: When the Attach method returns successfully, the attachment will affect the agent instances being attached. Both of them must be locked until the attachment operation ends. Similarly, God. World Agent List must be locked by agent creation code when adding a new agent instance to the list, and each agent instance must be locked during its state update.

```
public class CoreController  {
    public God God;
    public SimpleThreadPool STP;
    /// Configuration strings from agent types in DLL.
    public List<ConfigStrings> ConfigStrings;
    /// All loaded DLL classes
    public ArrayList Classes;
    /// Types of agent classes: for xml save/load
    public Type[ ] AllTypes;
     /// XML save/load functions
    public XMLController XMLController;
    public CoreController ( )
    public ArrayList GetAllTypesFromDLLstring (string dllName)
    public ArrayList GetAllTypesFromClass (string dllName,
                                           string className)
    public object RunClass (string dllName, string className,
                            string methodName)
    public object CreateDLLInstance (string className,
                                     params object[] args)
    public void EnableMarkerAnimation()
    public void DisableMarkerAnimation()
    public void DeselectMarkers()
    public void StartThreadPool (int ThreadsNum,  int IdleTimeout,
                                 int ExecuteTime)
}
public CoreController ()
{
    this.God = new God();
    this.XMLController = new XMLController(this);
    ABDiSE.Model.Environment SimulationWorld =
                        new ABDiSE.Model.Environment ( … );
    God.AddToEnvironmentList (SimulationWorld);
    string DLLName = "AgentDLL";
    Classes = GetAllTypesFromDLLstring (DLLName);
    ConfigStrings = new List<ConfigStrings>();
    for (int ii = 0; ii < Classes.Count; ii++)
    {
        ConfigStrings configStr = (ConfigStrings)RunClass (
                                  DLLName,
                                  Classes[ii].
                                  ToString (),
                                  "SetDefaultConfigStrings“ );
        ConfigStrings.Add(configStr);
    }
}
```

Fig. 5. Parts of class Core controller and core controller constructors.

Lock contention is reduced and time order is preserved by dividing each simulation step into two sequentially executed parts. In the first part, the update methods of all activated joined agent instances are executed. The update method of each joined agent updates its state and may create new agent instances. The update methods of all agent instances that are not of joined agent types are executed in the second part of the time step. As stated earlier, the method first tries to find suitable agent instances nearby and attach to them if possible. When attachment of an agent instance succeeds, there is no need to update the state of the agent instance. In other words, the state of any agent instance is updated only when the instance does not attach to another agent instance. At the end of the second part of a step, when update methods of all agent

instances have completed, the God instance executes ClearDeadAgent method to remove all dead agent instances.

VII. Case Studies

In case studies for purposes of assessing the usefulness and usability of the model elements and tools, we constructed from available model elements in the agent library several customized simulators of disasters at locations of well-known past disasters. They include a simulator of the devastating flood at Xiaolin Village in Taiwan caused by Typhoon Morakat in 2009 [25]; a simulator of an earthquake-triggered fire along Van Ness Avenue, San Francisco, CA, in the midst of the region hard hit by the 1906 San Francisco earthquake [26] and a simulator of frequent bushfires in New South Wales, Australia [27]. The screen shot shown in Fig. 2 was taken during the early part of a simulation run of the flood simulator. (Again, the small squares and circles in Area B represent houses/buildings in the affected area.)

Similarly, the top part of Fig. 6 shows a screen shot generated by the San Francisco fire simulator, showing the start of the fire from a few buildings. The bottom part of Fig. 5 depict the output from the South Wales bushfire simulator, showing the quick spread of fire (in red) and smoke (in dark cloud shapes) from a few buildings and trees.

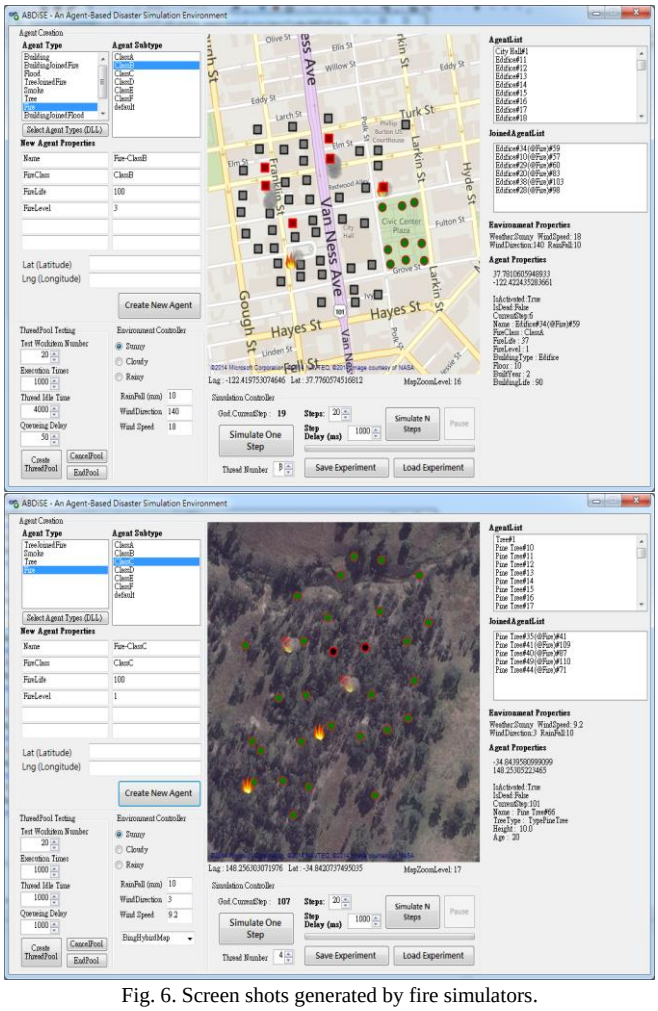

Fig. 6. Screen shots generated by fire simulators.

VIII. Summary and Future Work

The previous sections described the agent-based models and tools provided by the agent-based disaster simulation environment ABDiSE for construction and execution of the agent-based models to simulate several common types of natural disasters. Models available in the agent library of the latest version, ABDiSE 2.0, includes agent classes/DLL functions for modeling fire, smoke, flood, tornado, buildings, trees, and joined agents of them. In addition to model elements, the framework provides the user with several easy-to-use tools, including tools for agent creation, environment control, simulation control, and an interactive GIS map, which the user can access via main window of the GUI. XMLController is for saving/loading the state of agents in the simulation world at any point during a simulation run to be used as a checkpoint and as an initial state of simulation experiments to be done later.

Thus far, we have focused primarily on ABDiSE models and tools using which a user with can easily construct customized simulators of different scenarios for diverse types of disasters at different locales. Our experimentation has demonstrated that we have achieved this goal. The accuracy of the simulators requires significant improvement, however. This can be done by incorporating accurate and detailed fire and flood simulation programs in the update methods of joined agents. Indeed, the most important future work is to provide a wide selection of accurate simulation programs for various disasters.

ABDiSE 2.0 not only allows the user to import external simulators but also to add new agent types without having to modify and compile the source code of the framework: It will be straightforward to add agent types needed to model scenarios such as infectious diseases, terrorist attacks and major traffic accidents. It will also be straightforward to add agents that model people in different roles during disasters and thus make ABDiSE applicable for modeling and simulating SOPs in response to disasters [22].

The source code of ABDiSE 2.0 is released under the GPL license. The concept of agent-based model supported by the framework and ABDiSE code are easy-to-understand. A user willing to revise the source code can enhance the framework along many directions. For example, in the current version, the God.CheckAgentAttachment() method is invoked by every activated natural element agent during each simulation step. The method confines the search for agent instances suitable for attachment among attachable object agent instances. Consequently, multi-level attachment is not supported. (In other words, it is not possible for joined agents to attach to another agent and other joined agent.) This limitation can be easily removed by modifying this method and behavior rules in Attach methods to include joined agent instances as attachable objects without changing ABDiSE 2.0 system architecture.

References

[1] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," in *Proc. National Academy of Science*, May 14, 2002.

[2] J. M. Epstein *et al.*, "Combining computational fluid dynamics and agent-based modeling: A new approach to evacuation planning," *Plos* ONE, vol. 6, no. 5, May 2011

[3] D. Helberg and S. Balictti, "How to do agent-based simulation in the future: from modeling social mechanism to emergent phenomena, to interactive system design," SFI Working paper No. 11-06-024, 2011.
[4] S. Bandini *et al*., "Agent-based modeling and simulation: An informatics perspective," *Journal of Artificial Societies and Social Simulation*, vol. 12, no. 4, 2009.
[5] N. Gilbert, *Agent-Based Models*, Sage Publications, 2007.
[6] Comparision of agent-based modeling software. (2015). [Online]. Available: http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software
[7] E. Horner. (2011). Fire simulation with FDS. [Online]. Available: http://ejrh.wordpress.com/2011/04/14/fire-simulation-with-fds/
[8] Fire Dynamics Simulator (FDS) and Smokeview (SMV). [Online]. Available: https://code.google.com/p/fds-smv/
[9] Harrington Group. Fire Dynamics Simulator (FDS) and Smokeview (SMV) – Bringing Fire Analysis to Life. [Online]. Available: http://www.hgi-fire.com/blog/fire-dynamics-simulator-fds-and-smokeview-smv-bringing-fire-analysis-to-life
[10] M. Sebhat and T. Heinzer. The Development of an ArcInfo Interface to the National Weather Service DAMBRK Model. [Online]. Available: http://proceedings.esri.com/library/userconf/proc97/proc97/to600/pap581/p581.htm
[11] P. Bates, M. Trigg *et al.* Amazon modeling. [Online]. Available: http://www.bris.ac.uk/geography/research/hydrology/research/flooding/amazon/,
[12] LISFLOOD-FP. [Online]. Available: http://www.bris.ac.uk/geography/research/hydrology/models/lisflood
[13] L. Pawlowicz. High-resolution sea level rise flooding animations. [Online]. Available: http://freegeographytools.com/2007/high-resolution-sea-level-rise-flooding-animations-in-google-earth
[14] L. Pawlowicz. Animated sea level rise in manhattan. [Online]. Available: https://www.youtube.com/watch?v=RUNsV0ofX-s
[15] C. M. Macal and M. J. North, "Introductory tutorial: Agent-based modeling and simulation," in *Proc. the 2011 Winter Simulation Conference*, pp. 1451 - 1464, Dec. 2011.
[16] N. R. Jennings and S. Bussmann, "Agent-based control systems – why are they suited to engineering complex systems," *IEEE Control Systems*, vol. 23, no. 3, pp. 61-73, June 2003
[17] M. Wooldridge, "Agent-based software engineering," *IEEE Proceedings*, vol. 144, pp. 26-37, 1997.
[18] R. Allan. Survey of agent based modeling and simulation tools. [Online]. Available:http://www.grids.ac.uk/Complex/ABMS/
[19] Repast Suite. [Online]. Available: http://repast.sourceforge.net/repast_3/index.html
[20] Any Logic. [Online]. Available: http://www.anylogic.com/
[21] K. Mustaphaa, H. Mcheicka, and S. Melloulib, "Modeling and simulation agent-based of natural disaster complex systems," *Procedia Computer Scienc*e, vol. 21, pp. 148-155, 2013.
[22] C. Y. Wu, "Approaches to model people and SOP in disaster scenarios," MS thesis, Department of Computer Science, National Tsing Hua University, Taiwan, April 2015.
[23] GMap.NET Homepage. [Online]. Available: http://greatmaps.codeplex.com/
[24] T. L. Hsu and J. W. S. Liu, "Agent-based disaster simulation environment," Technical Report No. TR-IIS-15-004, Institute of Information Science, Academia Sinica, Taiwan, April 2015.
[25] T. Morakot. [Online]. Available: http://en.wikipedia.org/wiki/Typhoon_Morakot
[26] 1906 San Francisco Earthquake. [Online]. Available: http://en.wikipedia.org/wiki/1906_San_Francisco_earthquake
[27] 2013 New South Wales Bushfires. [Online]. Available: http://en.wikipedia.org/wiki/2013_New_South_Wales_bushfires

Tzu-Liang Hsu received his bachelor's and M.S. degrees in computer science in 2012 and 2014, respectively, from National Tsing-Hua University, Taiwan. He is currently working as a software engineer of Mediatek Inc, Hsinchu, Taiwan. His research areas include agent-based models and simulation and software framework design and development.

J. W. S. Liu received her B.S. degree in electrical engineering from Cleveland State University in 1959 and her Sc.D. in electrical engineering in 1968 from Massachusetts Institute of Technology. She is a distinguished visiting research fellow at the Institute of Information Science, Academia Sinica, Taiwan. She was an architect in the OS Base Core Technology Group of Microsoft Corporation from 2000 to 2004. Before joining Microsoft, she was a faculty member of the Computer Science Department at the University of Illinois at Urbana-Champaign from 1972 to 2000.

Her research interests are in the areas of real-time and embedded systems. Her recent work focuses on technologies for building user-centric automation and assistive devices and services and ICT technologies for disaster preparedness and response. She received the Achievement and Leadership Award of IEEE Computer Society, Technical Committee on Real-Time Systems in 2005; Information Science Honorary Medal of Taiwan Institute of Information and Computing Machinery in 2008 and Linux Golden Penguin Award for special contributions of Taiwan Linux Consortium in 2009.