

# VesselWorld and ADAPTIVE

Seth M. Landsman <sup>\*</sup>, Richard Alterman <sup>†</sup>, Alex Feinman <sup>‡</sup>, Joshua Introne <sup>§</sup>  
Dept of COSI, MS018  
Brandeis University  
415 South St  
Waltham, MA 02454

## ABSTRACT

The VesselWorld system is a same-time / different-place groupware system for cooperatively solving a problem requiring coordination. VesselWorld provides several tools for helping the players solve the coordination task.

In developing the VesselWorld system, a framework was developed which is designed to allow the building of further groupware systems. The framework allows for building systems which are scalable, easy to build, reusable and tailorable.

## Keywords

groupware, framework, group problem solving

## INTRODUCTION

The CSCW community is interested in groupware systems that support collective problem-solving and distributed activity. The decreasing costs of bandwidth and increase of telecommuters pushed for a demand for greater selection of products to support distributed computer-mediated same-time / different-place collaborations. Where before groupware applications were more of a research problem, increasingly it is part of the normal business of everyday work.

Development of collaborative and distributed groupware systems is expensive in terms of time and resources. Libraries have been developed within the CSCW community that potentially expedite and simplify the development processor. DISCIPLE [2] provides a replacement standard class library to transparently turn a single-user system into a groupware system. GroupKit is a library used for building groupware applications in tcl/tk. A daemon handles routing of data between components.[1]

---

<sup>§</sup>seth@cs.brandeis.edu

<sup>†</sup>alterman@cs.brandeis.edu

<sup>‡</sup>afeinman@cs.brandeis.edu

<sup>§</sup>introne@cs.brandeis.edu

The VesselWorld system, presented in this demonstration, is a same-time / different-place collaborative groupware system. In the VesselWorld system, three players participate in clearing a virtual harbor of toxic waste barrels. Different barrels require different equipment or more than one actor to handle. Others cannot be moved without the aid of a small barge. Such complications require complex coordination between the players. VesselWorld contains tools to help manage the coordination between users.

The development of the VesselWorld system has produced a framework for building groupware systems. The framework provides a way to build groupware systems which are scalable, easy to build, reusable and tailorable. This framework views a system as a collection of small components. It handles the task of organizing those components and facilitating the communication between them.

## ABOUT THE DEMO

This demo presents the VesselWorld system group problem solving system. During a session, the three people playing the VesselWorld system collaborate on separate machines, each playing the captain of a different ship, with different capabilities. The goal for a session is to clear the virtual harbor of barrels of toxic waste. There are different types of toxic waste, each requiring different capabilities, requiring different actors to participate in handling the waste.

The tug actor can move the small barges. She also has the capability to seal leaking waste and can determine what type of equipment is necessary to contain and lift a waste. The cranes have the capability to actually lift the waste. Waste that is large or extra-large need both cranes cooperating to lift it. Waste that is extra-large cannot be carried by any number of cranes, requiring that it be put on a small barge and then moved to the large barge

Planning and executing plans in the vesselworld domain is turn-based. Each player creates and submits a plan. When all players have submit a plan, the server will resolve the set of plans and return a new layout for the harbor. Failing an action (lifting something too large, for example) will cause the waste to start to leak.

The VesselWorld system provides different interface tools to handle the coordination and information management task

associated with solving the coordination problems that occur during a session. The windows the panels of information that VesselWorld has are :

**Control Center** The Control Center provides access to the other panels of information. Clicking on the Planning button, for example, brings up the planning panel. When a panel is changed that is not open or has focus, the corresponding button on the control center will flash

**Planning Panel** The Planning Panel presents all of the actor's current plans. The plan to be executed this turn is always at the top.

**Chat Panel** The chat panel allows the players to communicate with each other during play in an unstructured format. This is a simple chat room, similar to the one described below

**Info Panel** The info panel allows the player to get information about wastes in the world. Generally wastes contain information about what size they are (which determines how many actors are needed), how quickly they are leaking, which determines whether or not an actor which can seal the waste is needed, and what equipment is needed (determining which actor needs to be present and deployed)

**Strategy Panel** The strategy panel allows the players to cooperatively design a high level strategy for how they are going to complete the problem. The plans which are built also contain information as to whether or not they are pending, in progress or done.

**Object List Panel** The object list allows the players to put the properties and names of each waste in a central location.

**World Panel** The world panel is the interface into the problem itself. The actual affect of actions within the world panel depends on the current state the system is in. For example, in a planning mode, clicking would create new plans in the planning panel. In information mode, clicking would query the object for new information.

## THE ADAPTIVE CLASS LIBRARY

A system that is built using the ADAPTIVE library has three levels of implementation that the programmer needs to worry about. These are denoted by the three classes that the programmer extends in building a system. The classes are:

**System** The System object is the container which holds the collection of subsystems which make up the actual groupware system. This will handle the wiring of the initial system and handle the instantiation of components themselves. The Chat room below is a system

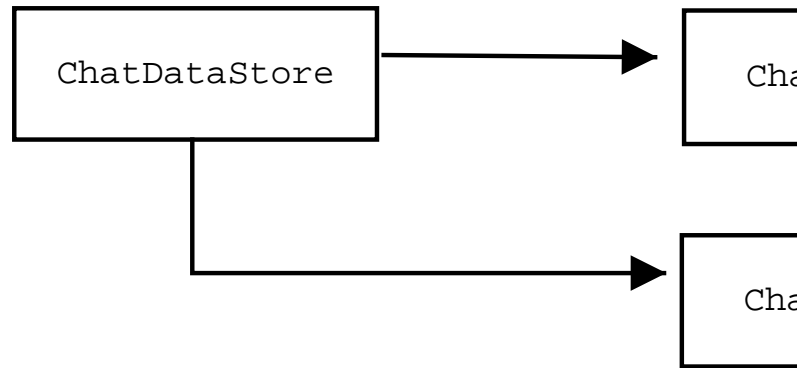


Figure 1: chat diagram

**Subsystem** The SubSystem object is the container which holds the ordered collection of components. The subsystem's main responsibility is to allow the passing of messages between components, which includes the registration and deregistration of actual components. The different subsystems, representing different users in the chat room are subsystems.

**Component** The components are the actual bits of code written by the programmer. These are pieces of useful objects ordered by purpose (denoted by being a part of the subsystem). The chat data store and chat user interface components are examples of Components.

An ADAPTIVE system is built as a hierarchy. A System contains several SubSystems. A SubSystem contains several Components. A simple chatroom might look like :

```

system -> chatdatacomponent -> chatviewcomponent (chat1)
                             |-> chatviewcomponent (chat2)
  
```

The System is built from the XML file, which describes the SubSystems and Components. A SystemManager instantiates the actual pieces of the system. For example, the ChatRoom system is built from :

```

<SYSTEM name=chatroom
  class=adaptive.system.BaseSystem id=0>
  <SUBSYSTEM name=chat1
    class=adaptive.system.BaseSubSystem id=0>
    <COMPONENT name=datastore
      class=sample.chat.ChatDataStoreComponent id=0/>
    <COMPONENT name=ui
      class=sample.chat.ChatUIComponent id=1/>
    </SUBSYSTEM>
  <SUBSYSTEM name=chat1
    class=adaptive.system.BaseSubSystem id=1>
    <COMPONENT name=datastore
      class=sample.chat.ChatDataStoreComponent id=0/>
    <COMPONENT name=ui
      class=sample.chat.ChatUIComponent id=2/>
    </SUBSYSTEM>
  </SYSTEM>
  
```

Communication within the system is handled by the framework exclusively. Each component needs to have a send() and receive() method, which are called by the SubSystem.

The code for the chat room follows. The SubSystem and System classes used are the default ones shipped with ADAPTIVE.

```

ChatDataStoreComponent :
public class ChatDataStoreComponent
  extends AdComponent {

  public ChatDataStoreComponent() {
    initializeDataStore();
  }

  public void receive(AdMessage message)
  throws AdException {
    if (message instanceof ChatMessage) {
      if (isNewMessage(message)) {
        send(message);
      }
    }
  }

  public boolean isNewMessage(ChatMessage message) {
    // returns true if this is a new message
  }

  public void initializeDataStore() {
    // initialize a new data store
  }
}

ChatUIComponent :
public class ChatUIComponent
  extends AdComponent implements ActionListener {
  public ChatUIComponent() {
    initializeUI();
  }

  public void receive(AdMessage message)
  throws AdException {
    if (message instanceof ChatMessage) {
      addMessage(message.getActor(), message.getMessage());
    }
  }

  public void actionPerformed(ActionEvent evt) {
    send(new ChatMessage(getActor(), getMessage()));
  }

  public void addMessage(String actor, String message) {
    // add the message to the ui
  }

  public String getMessage() {
    // get the message from the ui
  }

  public String getActor() {
    // get the actor name that this component represents
  }
}

ChatMessage :
public class ChatMessage extends AdMessage {
  public ChatMessage(String actor, String message) {
    // store actor and message
  }

  public String getActor() {return actor;}
  public String getMessage() {return message;}
}

```

A system can be defined as an XML document and set of Component classes. Once a collection of Components exists a new system can be built from old components by specifying an XML document. ADAPTIVE will provide a way of determining compatibility between Components. A specific

component might not accept the same messages that another might send, or might be expecting certain data to exist in a message. By ensuring that two Components are compatible, a system can safely be built out of existing collections of Components.

The ADAPTIVE framework also allows for complete tailorability of the system structure. Because the Components are self contained, new components can be added without affecting existing components. Also, current components can be replaced by new components, without disrupting the system itself.

## CONCLUSIONS AND FUTURE WORK

The ADAPTIVE framework provides a way to build reusable and robust groupware systems. VesselWorld, which this framework grew out of, is one such example.

Currently under development is a suite of tools to analyze the data which the VesselWorld system collects during runtime. In this suite is a VCR tool which will allow the play back of data collected using the very slightly modified SubSystems from the VesselWorld system. Because the VesselWorld components and subsystems were designed to be reusable in this fashion, the task of writing this system is much reduced.

Further, there are tasks to be done on top of ADAPTIVE, which would make implementation of new systems on top of ADAPTIVE easier and more robust. This would include writing a library of Components which are generally useful to ADAPTIVE-based systems. These Components would include items like event logging and general data storage.

## REFERENCES

1. M. Roseman and S. Greenberg. Building real time groupware with groupkit, a groupware toolkit. *ACM Transactions on Computer Human Interaction*, 3(1), March 1996.
2. W.Wang W. Li and I. Marsic. Collaboration transparency in the disciple framework. In *Proceedings Of ACM International Conference on Supporting Group Work (GROUP'99)*, pages 326–335, Phoenix, AZ, November 1999. ACM.

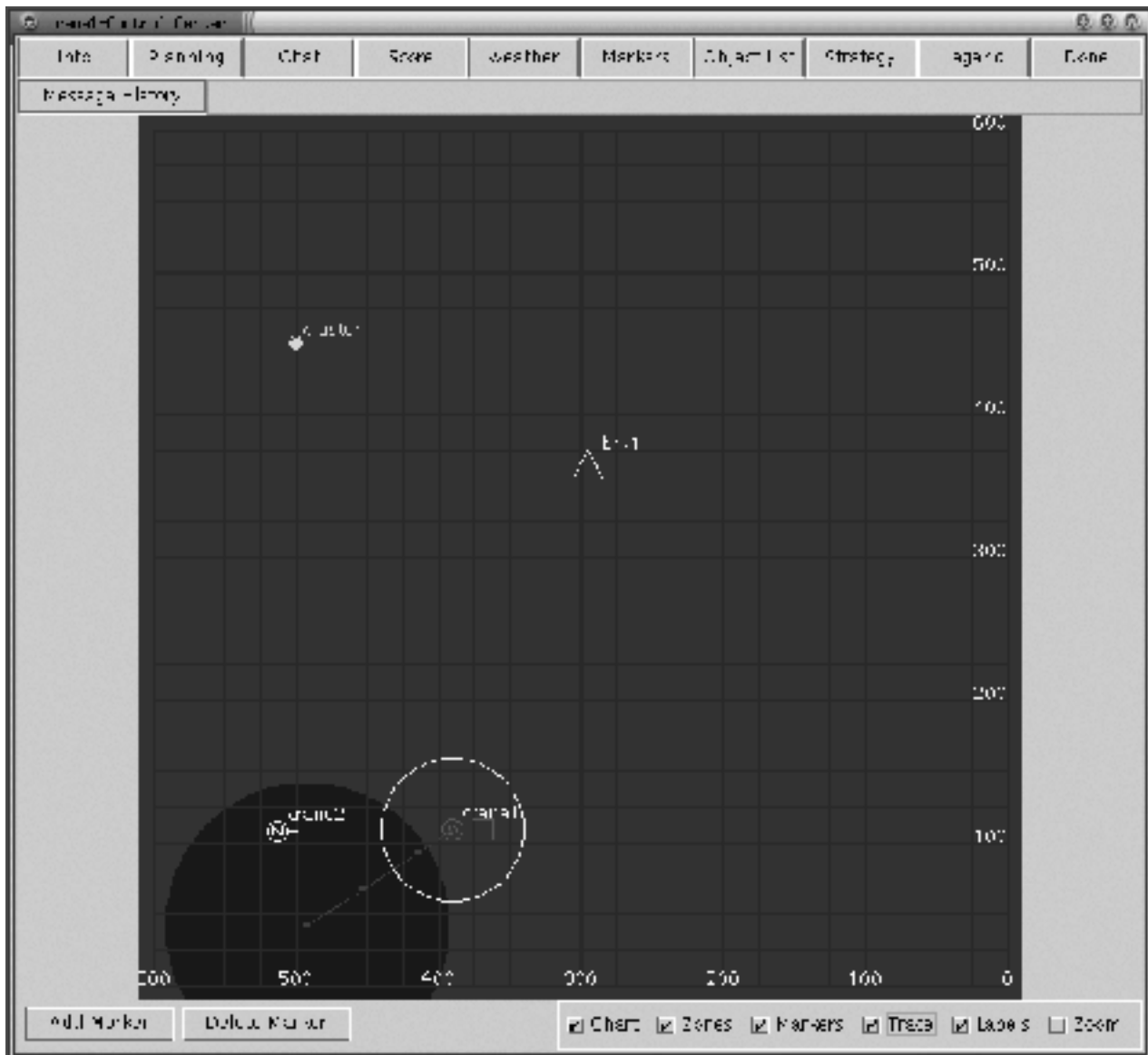


Figure 2: The VesselWorld Control Center