ERG4920CR Thesis II

Event Server with Event Processor on Mobile Device

BY

Hui Ka Hung

A FINAL YEAR PROJECT REPORT
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF INFORMATION ENGINEERING
DEPARTMENT OF INFORMATION ENGINEERING
THE CHINESE UNIVERSITY OF HONG KONG

May, 2005

Content

1. Abstract

In this thesis, the implementation of the Event Server and the Mobile Event Processor will be demonstrated. Practical uses of the Event Server and the Mobile Event Processor, and a comparison with other existing solutions are discussed to illustrate their potential in solving problems faced by adaptive enterprises. The property of being dynamically configurable makes it particularly suitable for adaptive enterprises.

2. Introduction

In the Information Age, information is being codified and distributed in an amazingly fast pace. To enterprises, this means that uncertainty becomes the only sure thing and the old "make-and-sell" model [1] collapses. In this model, all changes are predicted and detailed plans are made beforehand to cope with those potential changes. However, in information age, this model faces the following challenges:

- ✓ Customers change their preferences very quickly, and market for the planned products may disappear. Competitors who offer new and improved products in time will grab the new market.
- ✓ New technology evolves rapidly, competitors may outperform by using those new technology.
- ✓ Rules are always changing and players come and go. This changing environment makes the planned strategy broken down and the planning cost and effort wasted.

Therefore, enterprises must adapt to changes in order to survive, embracing the "sense-and-respond" model. The behaviour of this model is: [1]

"A sense-and-respond organization does not attempt to predict future demand for its offerings. Instead, it identifies changing customer needs and new business challenges as they happen, responding to them quickly and appropriately, before these new opportunities disappear or metamorphose into something else."

In this new model, the following qualities should be equipped:

- ✓ It should be able to sense a wide scope of customers' preferences and have accurate analysis on them quickly. Life cycle must be shortened.
- ✓ It should enable employees to make decentralized decisions to cope with the emerging opportunities in time. In other words, the employees are given a set of directions and constraints, and the employees are empowered to make decisions within their own context.
- ✓ Besides connecting to customers, it should also connect to their partners closely to make co-operative production strategies, so as to achieve the maximum economy of scales.
- ✓ It should always adapt to roles and boundaries changes, both inside and outside the enterprise.
- ✓ Co-ordination must be stressed: while employees are empowered to make decentralized decisions, they should co-ordinate to get a unified view of the environment and key processes.
- ✓ Parallel processing of the explosive incoming information is a must.
- ✓ Services become modular and are dynamically combined to respond to specific needs of customers.

The biggest challenge here is speed: how to adapt to changes within the shortest time, how to absorb information before they go outdated, how to navigate the channels that deliver the hottest news, *etc.* In order to solve all the problems brought by the extensive use of information technology, more information technologies come to rescue.

This thesis will be divided into the following parts. In Section 3, some background information about collaboration concepts of distributed systems will be introduced. In Section 4, the implementation of the Event Server and the Mobile Event Processor will be discussed. In Section 5, how the Event Web, which is based on the concept of event-based collaboration, and equipped with Mobile Event Processor and Event Server, can help overcome the difficulties faced by adaptive enterprises will be demonstrated. Comparisons with other popular solutions like Web Services and remoting will also be shown.

3. Background

When different entities, both inside and outside the enterprise, are connected together, a distributed system is formed. A distributed system can be generally defined as a set of related computing entities working together to accomplish a common function, and are connected through a communication scheme. The collaboration concept of a distributed system can be characterized by the following questions:

- ✓ How do entities communicate with each other?
- ✓ How are different types of requests identified?
- ✓ How do entities discover each other?
- ✓ What are the roles of different entities?

The three popular collaboration concepts of distributed systems are instance-based, service-based [3] and event-based. [8] Their differences are characterized in Table 1.

These collaboration patterns of distributed systems have many impacts on their performances and usages. For example, in event-based collaboration, or Event-Driven Architecture (EDA), entities communicate by sending events which contain pieces of state or state history of a system. Any object interested in the event will act upon reception, making the actions asynchronous and concurrent. As a result, users can be alerted in real-time for notifications or decision-making. Responses of users can also be turned into events so that servers equipped with event-dispatching capability, which are labelled as Event Servers, can immediately act on. Real-time actions can thus be carried out and latency is minimized. Besides, Mobile Event Processors, a piece of software specially designed for mobile devices to process events, can help correlate, filter and divide the events scattered throughout the network into hierarchies. This helps to save bandwidth and resources, match and reconcile information from different sources, and enable high-level decision-making based on low-level event streams.

	Instance-based	Service-based	Event-based
Communication format	Remote procedure calls - one-to-one - platform dependent	Service calls - one-to-one - can be platform neutral	Event streams/timeout - many-to-many - platform neutral
Request Identification	Objects interfaces	Service contracts	Event schema
Discovery	Manually fixed - no dynamic change allowed	Service Directory - no dynamic change allowed	Event Subscription - dynamic change supported
Roles	Fixed roles 1. Object provider 2. Object consumer	Fixed roles 1. Service provider 2. Service consumer	Dynamic roles Agents: produce and consume events at the same time
Example	.NET remoting	Web Services	Event Web

Table 1: Comparison between instance-based, service-based and event-based collaboration.

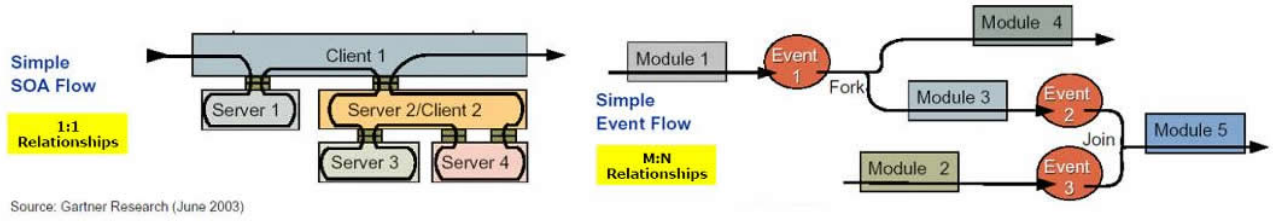

Figure 1: EDA and Service-Oriented Architecture (SOA), an example of service-based collaboration.

4. Implementation

4.1. Event Server

4.1.1. Introduction

The Event Server is a server equipped with event-dispatching capability. It is built in such a way that multiple Event Chains can be executed concurrently. An Event Chain is defined as a mixture of state transition diagrams and event-driven functions, and is used to control the behaviour of the Event Server. By introducing Event Chains, users can specify the system behaviour in either a stateful or stateless manner, and the relations between events can be clearly shown. Besides, users can group events by their nature (*e.g.*, HK weather events, HK accident events group under HK), enabling event subscriptions to be done in a loosely-coupled manner. All these introduce more expressiveness for describing the behaviour of the Event Server, thus increases the usability and user-friendliness of the system. In addition, the Event Server is able to consume Event Chains on-the-fly, therefore it is possible to change the system behaviour at runtime. By taking reference to the Distributed Agent System [4], Agents are introduced to monitor any arbitrary current situations of particular interest. Different Agents can run at the same time, and it is possible for one event to trigger multiple actions on multiple agents.

4.1.2. Agent and Event Chains

An Agent contains a set of variables which represents the current situation of particular interest. An Agent can be attached to multiple Event Chains, with each of them representing how an Agent reacts towards certain types of event. Upon arrival of a specific type of event, the corresponding Event Chain will be invoked to update the variables in the Agent and to generate an output event.

An Event Chain consists of two parts. The first part is a state transition diagram: upon event trigger, if the current state of the Agent maps any of the states in the diagram, the corresponding transformation rules will be executed. The state of the Agent will be changed as a consequence of event trigger. The second part is a set of event-driven functions, which will be executed whenever there is an event trigger of the corresponding event type, so operations can be carried out in any state. In this way, how an Agent reacts towards event trigger can be specified in either a stateful or stateless manner. A graphical representation of the Event Chain is shown in Figure 2.

Upon event trigger, the Event Chain may instruct the Event Server to send an XML file back to the event source. In this situation, the Event Chain must specify the destination to send the file, and the destination must be one of the event sources registered to it; otherwise, errors will be reported and the execution of the Event Chain will be terminated. If the destination specified is “all”, then all the event sources registered will be used as destination.

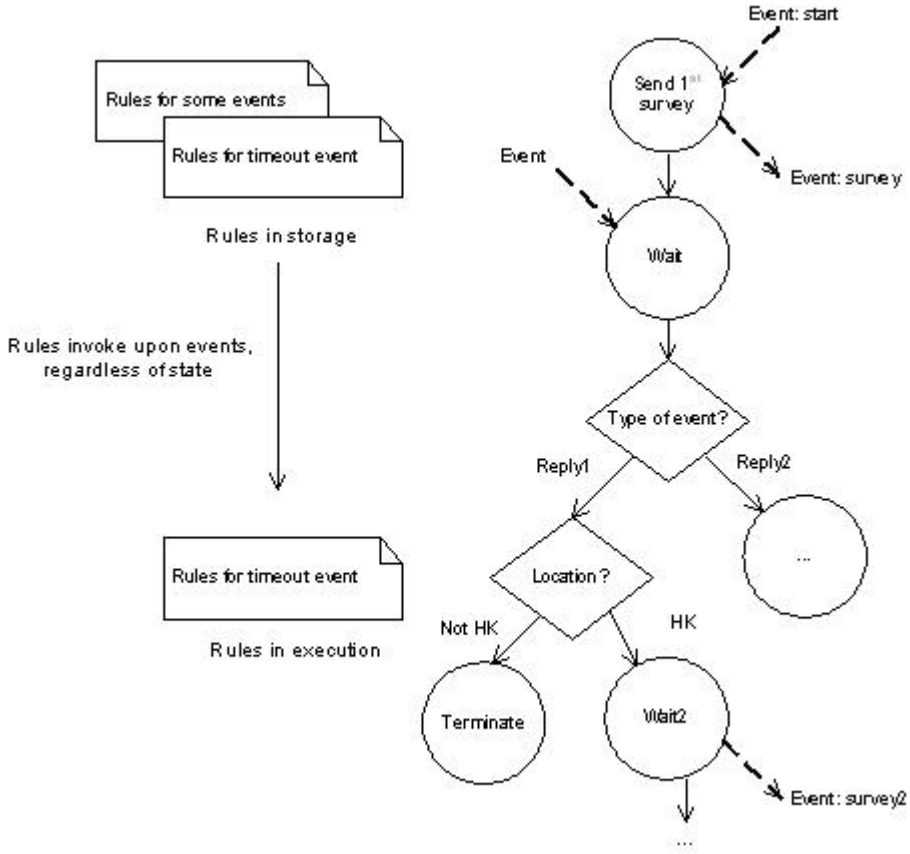

Figure 2: Graphical representation of Event Chain - rules execution depends on both states and events.

4.1.3. Processors

How Event Chains are related to Agents and how event trigger leads to execution of Event Chains are managed by Processors.

Processors are identified by the nature of events they are interested in. They have a list of event sources which provide events of similar types. Agents can subscribe events by adding themselves to Processors with the corresponding Event Chains specified, in which the Processors will invoke the Event Chains upon event trigger. A Processor only pays attention to the events that are of its interest. Thus, the event-driven functions in the Event Chains must be relevant to the events that are of the Processors' interest; otherwise, the Event Chains will never be executed.

Multiple Event Chains can be executed concurrently on an Agent by attaching it to multiple Processors, with each Event Chain representing a group of related events that are of the Agent' interest. Figure 3 shows this scenario.

In addition, subscription to an event can be easily terminated by detaching the Agent from the Processor. This does not affect other event subscriptions since they have different Event Chains and do not share the same set of state variables. This is also the same for adding event subscriptions, which can be done by simply adding the Agent to the Processor with the Event Chain specified. An Agent can be easily added to or removed from a Processor since the variables specified in an Agent is independent of any event sources that it is interested in, except for the variables which represents the current state of each Event Chain.

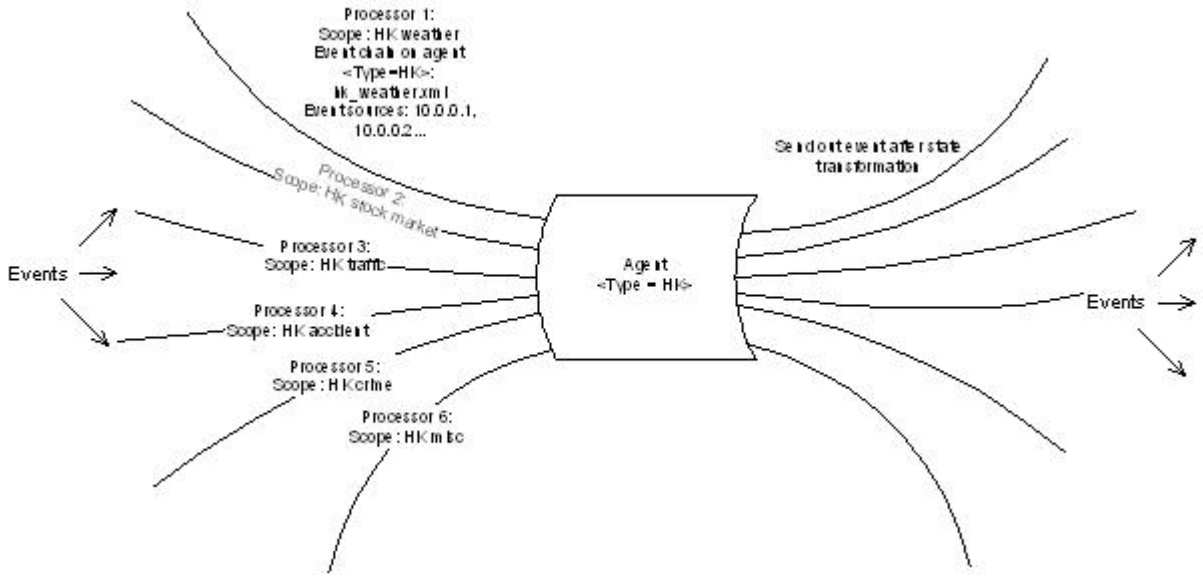

Figure 3: Each Agent can be handled by multiple Processors.

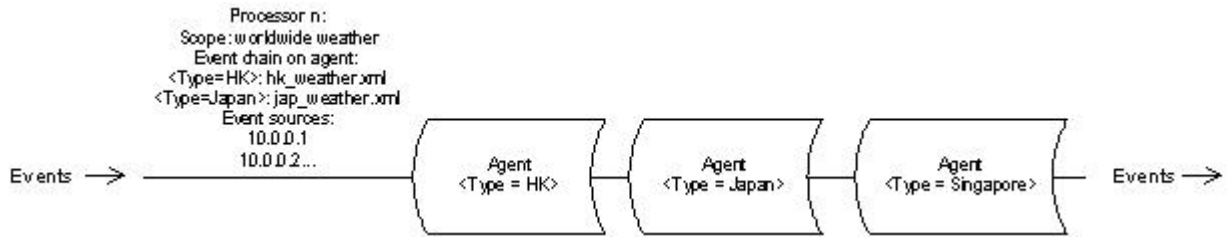

Figure 4: Each Processor can handle multiple Agents concurrently.

What's more, each Processor can manipulate multiple Agents, so that a single event may trigger transformations on different Agents, with the expected responses of the Agent stated in different Event Chains. A graphical illustration is shown in Figure 4.

The behaviours of the Event Server can be even more dynamic by requiring that the Event Chain to be dynamically configurable. This can be accomplished by using Web Services, and will be discussed in Section 4.3.

4.2. Mobile Event Processor

4.2.1. Introduction

The Mobile Event Processor is a piece of software specially designed for mobile devices to process events. Arrival of events invokes the Mobile Event Processor to process the events. Internally, its job is shared among Agents, and each Agent further delegates the job to Actuators. Therefore, in each processing cycle, a chain of responsibility [9] is formed. In this section, a detailed profile about Agents and Actuators, and some other related issues are discussed.

4.2.2. Processing Cycle of Creating New Agents

When an incoming message arrives, an Agent will be invoked to process the message. An Agent is an object that carries out operations on arrival of messages. An Agent can be viewed as a store of state information. For example, an Agent may store the previous answers made by a user when presented with a

survey. What an Agent can perform depends on its state and the content of the message to work on.

Agents are identified by their type and scope. The type of an Agent represents the operations that are expected for the Agent, while the scope of an Agent is the event requesting for the operations. For example, a voting system and an online survey both generate events to the Mobile Event Processor. Both events require it to display user interfaces. Due to the similar nature of the actions required, these two events may be handled by Agents of the same type, say, "UI". Since the actions are triggered by two different events, these two events should be handled by two Agents of different scopes, say, "survey" and "vote".

Due to the nondeterministic nature of event-based collaboration, it may happen that the Agent with the corresponding type and scope may not exist when a message arrives. In this case, a new Agent must be created in order to consume the event.

If the scope of the Agent does not exist but the type exists, *i.e.*, the initial state file for the Agent of this type exists in the local file system, then a new Agent can be created instantaneously and the initial state file will be read to initialize the state of the Agent. The state information is then stored in a table in the Agent. Then the message can be processed by this newly created Agent.

If both the type and scope of the Agent do not exist, a message will be sent to the Event Server to request the initial state file of the Agent. Then the message will be queued in an internal queue. The purpose of this internal queue is to store the messages that do not have corresponding Agents to process it. When the Mobile Event Processor receives the initial state file from the Event Server, it saves the file into the local file system, and the Agent of this new type can be created. The messages in the internal queue will be re-processed.

Through these operations, events of arbitrary type and scope can be consumed by the Mobile Event Processor on-the-fly. Event buffering is also possible to store the events that cannot be handled currently.

4.2.3. Processing Cycle of Consuming Incoming Events

When an Agent is created, it reads the corresponding initial state file and gets a copy of the default Actuator. An Actuator is an object that carries out operations on messages on behalf of the Agent which owns it. The Actuator does not carry any state information. Before any operations are carried out, the Agent feeds its own state information to the Actuator to initialize the Actuator and passes the message to the Actuator. Then the Actuator performs operations on the message according to the states of the Agent.

The action of the Actuator can be classified into two types: synchronous and asynchronous:

- ✓ In synchronous operations, after receiving the state information and the message, the Actuator will process the message, update the state, return the state to the Agent and compose another message immediately which is to be returned to the Agent.
- ✓ In asynchronous operations, the Actuator will process the message and wait for user input. After collecting user input, the Actuator will update the state, return the state to the Agent and return the user input in the form of a message to the Agent.

4.2.4. Multithreading Issues

There is exactly one instance for each kind of Actuator. The reason for this is obvious: Consider there is a kind of Actuator that can construct and display user interfaces. If there are two Actuators of this kind that are active simultaneously, these Actuators may simultaneously display two different user interfaces to

users. This may be problematic to users as they may not know which interface they are now working with.

Even there is only one instance for each kind of Actuator; the problem is not completely solved. Consider the case that during the processing of the first message, the second message of the same kind arrives. The corresponding Agent will be chosen and this Agent utilizes the Actuator which is processing the first message. This Agent feeds its own state information to the Actuator, which overwrites the original state information. The processing of the first message may be corrupted as a result.

Processing two or more events simultaneously by the same Actuator must be prohibited. The approach adapted to solve this problem is that, when a second event of the same type arrives, the processing of the second message will be suspended before the stage of feeding state information, until the processing of first message is finished. This helps to maintain the consistency of state in the whole processing cycle.

4.2.5. State Transformation Issues

To transform the state of an Agent, one approach is to request the Mobile Event Processor to send the states of the Agent to the Event Server, process the state transformation by the Event Server, and return the transformed states to the Agent in the Mobile Event Processor. The drawback of this approach is that the time incurred may be too large. An alternative to do this would be to encapsulate the state transformation in an XML file, send the XML file to the Mobile Event Processor and instruct the Agent to transform it. This approach not only reduces the latency, but also consumes the business logic on-the-fly.

The state transformation is carried out by utilizing the Actuator which is specifically designed for transforming states. The XML file encapsulating the transformation rules only needs to specify the type and scope of the Agent, the name of the Actuator, and the conditions and methods to transform states.

A Logic Actuator is used to carry out any state transformations. The type and scope of the Agent is used to choose the Agent to perform state transformation. The name of the Actuator is needed to override the default Actuator used by the Agent. The methods of transformation are “insert”, “delete” and “update”. New values of the state variables can be specified as constants, or arithmetic operations on current state variables. The conditions for transformation are basically comparison between state variables and/or constants. The comparisons are classified into six groups: “eq” (equal), “ne” (not equal), “le” (less than or equal to), “lt” (less than), “ge” (greater than or equal to) and “gt” (greater than). These comparisons can be linked up logically by using “not”, “and”, and “or”, with “not” the highest precedence and “or” the lowest.

To specify the rules for state transformation, the only thing the Event Server needs to know is the name of the state variables that the Agent owns. It is not necessary for the server to know the values of the variables before it can generate any rules.

The state information inside an Agent, together with the rules of state transformation specified in the XML file, helps emulate a state machine. The actions carried out by an Actuator can be controlled by the state machine, so the actual behaviour of any Actuator (and hence the behaviour of any Agent) can be dynamically configured.

If there is any syntax error, an exception is thrown and the previously performed state transformations (if any) are rolled back to ensure the consistency. This is performed by making a deep copy of the table before carrying out the state transformation. When error occurs, the deep copy will replace the current table.

4.2.6. State Synchronization between Server and Event Processor

When the Event Server requests the Agent to return its states to the Event Server, the Agent will use the state information in the table to compose an XML file and return it to the Event Server. The Event Server can request the state information of any Agent at any time. The state information requested may be used for synchronization to ensure that both the Event Server and the Mobile Event Processor have the same state.

Synchronization between the Event Server and the Mobile Event Processor helps reduce the need for buffering messages in the Mobile Event Processor. Consider the case that the Mobile Event Processor does not have any Agent that is capable of constructing user interfaces. Sending many messages related to constructing user interfaces may cause the Mobile Event Processor to buffer all these messages. The querying of state information by the Event Server enables it to decide to send the initial state file first to the Mobile Event Processor to create the Agent, and then send the messages of user interface; or not to send any messages of user interface to the Mobile Event Processor at all. This helps reduce the need for buffering messages and the time needed for processing messages.

4.2.7. Database Management Issues

A Database Actuator is designed to handle messages that encapsulate serialized datasets in XML. Through this actuator, the Mobile Event Processor can view and modify datasets of arbitrary schema, thus enabling the Event Server to push database objects. This Actuator have simple interfaces as well as those popular data manipulating functions, *e.g.*, adding and deleting rows, changing information of existing rows; viewing data with sorting and filtering; keeping track of modified history. The user input will be checked against constraints and the expression syntax will be verified. Thus error checking is localized in the Mobile Event Processor, which reduces the workload of the Event Server. The dataset after manipulation will be sent back in Diffgram. [7] A Diffgram includes both original and current values of each row in the table. As a result, it always provides a way for the Event Server to keep track of the changes if it wishes. The Diffgram will also be stored in the corresponding Agent as history. Therefore, the Database Actuator enables the Mobile Event Processor to integrate with data-centric distributed applications.

Furthermore, this Actuator provides a user interface for specifying events as datasets, which can be easily serialized into XML. In other words, this Actuator can be further utilized for event triggering between the Event Server and the Mobile Event Processor. For example, users can specify the Event Chain as a dataset and sent it back to the Event Server in XML. As a result, users are able to create new Event Chains or to modify executing Event Chains at runtime and at anywhere, and change the behaviour of other Mobile Event Processors on-the-fly. Therefore, it is possible for mobile devices to move towards the centre of EDA in helping event triggering.

4.3. Specification of Event Chain: Web Services

To provide greater flexibility, all Mobile Event Processors are able to change the behaviour of the Event Server at runtime, so as to react towards sudden circumstances happened in the local area. In this way, the Event Server is equipped with the maximal adaptive power to cope with environmental changes. Also, both the Event Server and the Mobile Event Processor have the power to invoke creation of new Event Chains. The newly created Event Chains can be used immediately so that the behaviour of the Event Server can be configured on-the-fly. In our implementation, how the Event Server requests the Mobile Event Processor to

create an Event Chain for it is demonstrated. This approach involves more interactions between the Event Server and the Mobile Event Processor, which demonstrates the following capabilities:

- ✓ Using events to communicate.
- ✓ How the Event Server functions based on incoming event content.
- ✓ How a generic event can be used in different situations.
- ✓ Providing a way for the Mobile Event Processor to alter the behaviour of the Event Server at runtime, so as to cope with the dynamic environment.
- ✓ How Actuators in the Mobile Event Processor co-operate together to perform complex operations.
- ✓ How different event sources contribute to compose an Event Chain.

4.3.1. Creation of new Event Chain

In order to make the Mobile Event Processor create an Event Chain, a new Actuator that has the Event Chain schema embedded is needed. However, the Event Chain schema may change from time to time. Thus every time the schema is changed, the Actuator needs to be re-implemented. Besides, if we create new Actuators when we require new actions, the mobile devices will be overloaded. Therefore, actions requiring constant changes are not implemented in the Mobile Event Processor. Instead, Web Services [6] are invoked to process the requests. As a result, only one Actuator is needed, and it is responsible for invoking Web Services only. Together with the Web Services Actuator, the whole structure of the Mobile Event Processor is shown in Figure 5.

4.3.2. Web Services Actuator and Web Services Server

The Web Services Actuator should be able to accomplish any actions that other Actuators cannot finish; in particular, it should be able to handle operations that are unknown until runtime. It is desirable that the Web Services Actuator can choose the right web method to consume based on the requests during runtime. However, this is not possible: the location of Web Services server and the web methods consumed has to be fixed before compilation.

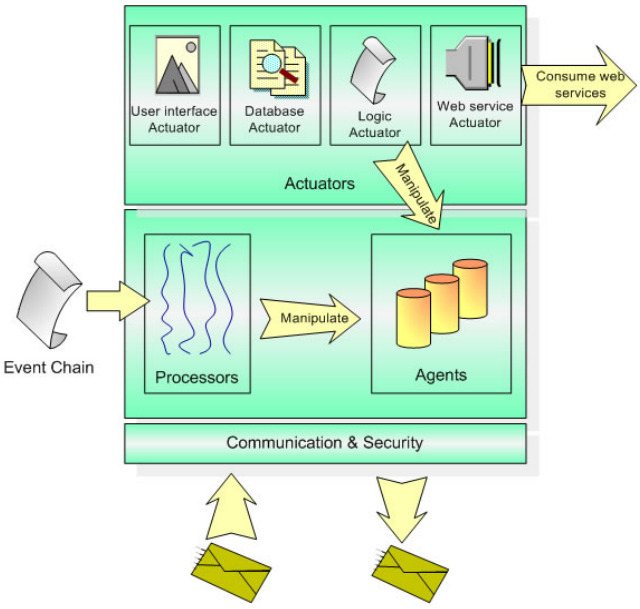

Figure 5: The whole structure of the Mobile Event Processor.

As a result, the Web Services server that provides the web methods to the Web Services Actuator needs special design. First, it should co-operate with the Event Server to find out what kinds of services it needs to provide for the Mobile Event Processor. Second, the single web method, which is called by the Actuator for all requests, should be able to multiplex those requests and invoke the right component to handle them.

In other words, the web method called by the Actuator acts as a gateway to the Web Services server. When a request string is passed to the web method, it invokes the corresponding module to handle the request. This is done by having a "Method factory", [9] which returns an object to process the request. In addition, the state of each request is cached on an application basis. This is to allow future requests to be processed based on state history. In the case of creating Event Chains, the Mobile Event Processor only need to send incremental updates to the Web Services server and the Web Services server will return the entire Event Chain based on the state history and the updated information. As it saves states on an application basis, it allows different sources to work together in creating the same Event Chain.

4.3.3. Mechanism of producing an Event Chain

In the implementation, the Event Server sends request to the Mobile Event Processor for a new Event Chain. Each time the Event Server only requests instruction for a single state or a single event. This is because requesting instruction for multiple states or events are too complicated, and the time it takes would be too long for regular mobile device applications. Sending too much information to a mobile device at one go is simply too overwhelming. Furthermore, the Event Server may change future requests to the Mobile Event Processors based on new instructions, to allow the creation of Event Chains in a more dynamic manner.

After each time the Mobile Event Processor completes the survey in stating new instructions, the Event Server can request the Mobile Event Processor to create the corresponding Event Chain, in which the job is delegated to the Web Services server. The web method caches the previous state information of the Event Chain, such that the Mobile Event Processor only needs to send the update of the Event Chain, and the Web Services Server will return a complete one. In addition, the state object is cached on an application basis, so the history of the composed Event Chain is shared by all Mobile Event Processors. As a result, an Event Chain can be formed as a result of co-operation among multiple Mobile Event Processors. In order to reuse the "instruction survey", the Event Server can change the content of the survey each time according to the guideline in the Event Chain. In addition, the algorithm in creating the Event Chain is intelligent enough to create the optimal chain structure, *e.g.*, removing unnecessary nodes, checking syntax, *etc.*

4.4. Network Protocol

Two issues are concerned in the interactions in the network protocol: the instability of the connection between Mobile Event Processor and the Event Server, and the asynchronous socket I/O.

4.4.1. Connection Instability

To handle the problem of instability in connection, a separate channel is maintained to "query" the Mobile Event Processor. The Event Server keeps an "active list", which contains the users that are currently connected to the Event Server. To "query" the Mobile Event Processor means that the Event Server sends a control message ("SYN") to the Mobile Event Processor. If the Mobile Event Processor is active, it will respond by returning the received control message. On receiving the same message, the

Event Server will mark this Mobile Event Processor as “active”. After a predefined period of time, say, ten seconds, the Event Server will check the “active list” again. If the Event Server finds that any Mobile Event Processor is not marked as “active”, the sockets corresponding to the “inactive” Mobile Event Processor will be closed. As a result, a data channel and a control channel will be consumed by a single Mobile Event Processor.

Another problem associated with the instability is that the IP of the Mobile Event Processor may change after sudden disconnection and re-connection. To identify the Mobile Event Processor, the protocol requires the Mobile Event Processor to send its own username and password to the Event Server. The Event Server maintains a list of valid usernames and their corresponding passwords in a dataset. For security reason, the passwords are encrypted by using the hash algorithm SHA1. When the Mobile Event Processor connects to the Event Server, it submits its username and password for authentication. The authentication process is done by traversing the dataset. Upon authentication success, the Event Server will send a control message (“SYN”) to the Mobile Event Processor to notify the success in authentication, and the Event Server will register the user as “active”; otherwise, the Event Server will send another control message (“FIN”) to the Mobile Event Processor to instruct it to close the connection.

4.4.2. Asynchronous Socket I/O

When the Event Server is started, it creates two sockets: one for accepting connections in data channel; the other for control channel. The Mobile Event Processor tries to connect to both data channel and control channel. After authentication, data transfer can be started at any time. Data transfer can be initiated by either side. All socket I/O calls involved are asynchronous. To operate in asynchronous mode means that the application uses threads in the system thread pool to process the following: [5]

- ✓ Connecting to remote hosts.
- ✓ Accepting incoming connections.
- ✓ Sending data to remote hosts.
- ✓ Receiving data from remote hosts.

When an asynchronous operation is started, the system thread will be running in the background to perform the operation. When the operation is finished, the completion routine specified will be invoked by the system thread to do any post-processing or issuing other asynchronous socket I/O calls.

5. Case Study

In this section, scenarios that may be faced by adaptive enterprises are discussed. The performance of Event Web and other solutions will be evaluated and compared.

5.1. Supply Chain Management

5.1.1. Connecting Nodes outside Boundaries

Scenario: In a supply chain, enterprises of different types have to be connected together: material provider, manufacturing sites, management firm, etc. It is much more difficult to compromise the communication pattern (platform, format, timing, etc.) among them. How should they be connected to achieve the maximal flexibility?

Current Solutions: The most effective way to connect enterprises using different platforms is by Web

Services. However, service contracts [3] for Web Services are rather restrictive: the signature of the service must be comprised at both sides. As a result, any changes must be compromised and done synchronously. In addition, the clients' states are kept at the server side, which increases the workload of the server and reduces scalability.

Event Web solution: Event schema is used to specify the type of interactions between entities. It is possible for either side to invoke system changes while the system is still running: the events are buffered while the event schema is not well understood. When either side changes the event schema, or a new schema is made to request for a new service, the system on the other side can buffer this unknown event and prompt for instructions. Moreover, the system can even prompt other nodes for instructions to process the unknown event. Thus when an entity encounters an unknown interaction type, it can ask for help rather then just 'die' as in Web Services (See Figure 6). This is very useful as program changes can be initiated programmatically and can be done asynchronously and seamlessly.

In addition, this event buffering mechanism is useful when invoking services while the environment is not well set. This is especially true when many entities are connected together and each entity depends on others to provide resources, in which synchronization of all the resources is hard to achieve. For example, the production site sends a request to the material firm to get more materials. However, the material firm still does not get permission from the management entity. As a result, the request events can be buffered and wait until the instruction from management entity arrives. This has loosened the timing constraint in connecting the entities.

Furthermore, Agent is introduced in the clients to keep its own states, which promotes loose coupling and reduces the workload of the server.

5.1.2. Unstable Topology: Nodes Come and Go

Scenario: In a supply chain, the entities always come and go, e.g., add more production lines, change manufacturers, etc. How can disturbance on the system be reduced?

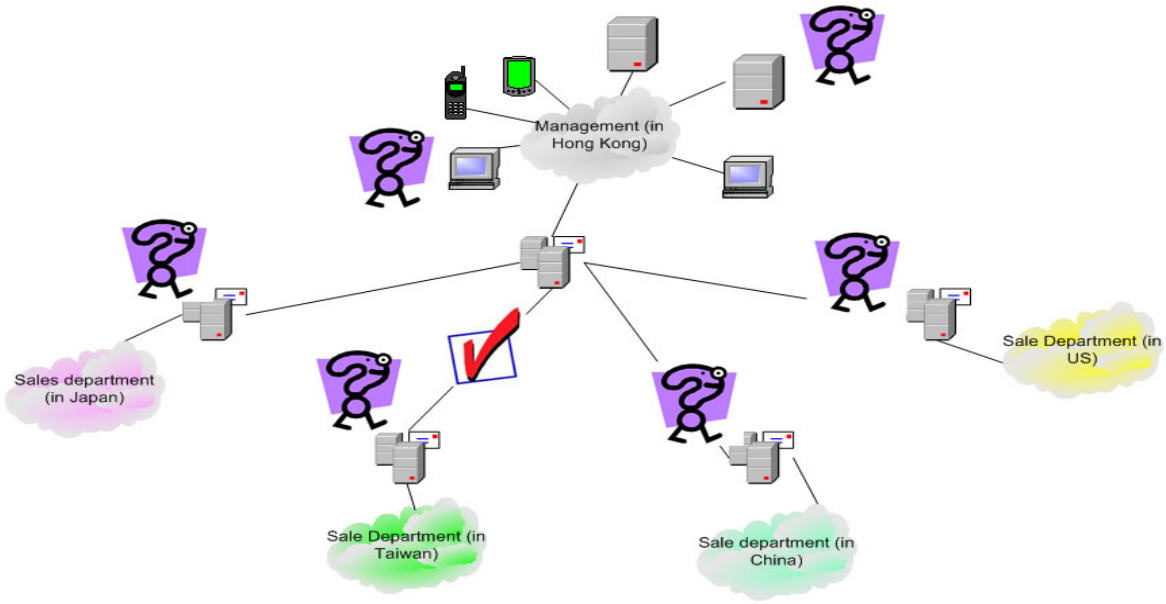

Figure 6: Asking for help when unknown entities are encountered.

Popular solutions: Web Services is the most popular solution to connect different enterprises since it can connect entities running on different platforms. In Web Services, the entities are connected through service binding. [6] First, the clients have to find out the contract through direct communication with the other side or through UDDI/DISCO. After that, the clients have to create a proxy locally to represent the remote service. All these have to be done statically: the proxy class must be compiled beforehand and there is no way to change it during runtime. As a result, whenever adding or removing services is needed, the system has to be re-built.

Event Web Solution: The entities are connected by sending events and they discover each other by event subscription. The idea of event subscription is similar to mail subscription: each entity shows its interests to different types of events by subscribing them, *e.g.*, production events, management events, *etc.* This subscription is done by registering itself to a directory similar to DNS. At the same time, each entity knows the nature of its outgoing events, *e.g.*, a production firm outputs production events. The event sender can lookup the directory to find the destination addresses of the event subscribers that are interested in its output events. The directory can be distributed and may form a hierarchy to enhance scalability. As a result, the event subscribers can receive the desired events without explicitly knowing who the senders are. In the example of adding new nodes in the supply chain, these new nodes can connect to others by subscribing desired events, identifying the nature of its output events and sending events to the appropriate event subscribers. As a result, no systems are required to re-build. A graphical illustration is shown in Figure 7.

This event subscription mechanism also helps consume services more efficiently: no explicit searching of services is needed. Instead, the right events come to approach the nodes if it has registered itself to the appropriate event sources. Also, the event senders do not have to wait for nodes to connect; it can seek event consumers actively from the directory. Furthermore, event subscription can be done on-the-fly.

5.1.3. Many-to-many instead of one-to-one

Scenario: In a supply chain, a production entity runs out of material may have a lot of follow-up actions: request for material replacement from the material provider, request for delivering delay, notify the management entity, etc. How can this production entity contact all the other parties efficiently?

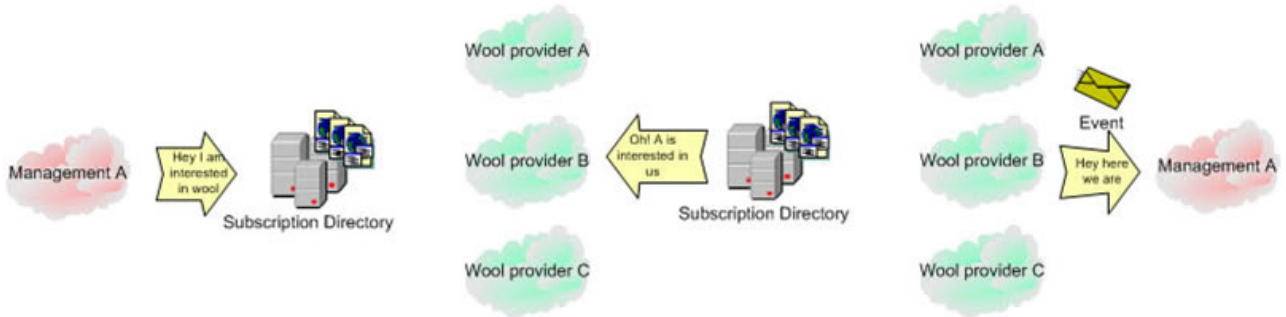

Figure 7: Event subscription.

(Left) Management A registered to the Subscription Directory.

(Middle) Wool Providers A, B and C lookup the Subscription Directory and find that Management A is interested in the events provided by them.

(Right) Management A received events from Wool Providers A, B and C.

Popular solutions: In remoting and Web Services, the nodes are connected by remote procedure calls or service calls. These are static and are one-to-one. As a result, the production firm has to contact its partners one by one. This is inefficient as the information is repeated and the bandwidth is wasted. In addition, these requests are done sequentially which introduces latency.

Event Web solution: In this example, the production firm sends the "out of material" event to the nodes it learns from the directory. The entities which have subscribed to these events, *e.g.*, the material firm, know how to react based on their own Event Chains. For example, upon "out of material" event, the material firm has to deliver more material; the management entity marks the period of requesting materials.

In other words, it is possible for a node to multicast its events to the nodes that have registered to the directory. Different entities can have their own interpretations of the same event by having different Event Chains. Since Event Chain can be changed on-the-fly, thus responses towards events can be altered when the environment changes. As a result, parallel processing of information is possible.

5.2. Emerging Markets

5.2.1. Add Instructions to Process Information at Runtime

Scenario: An incoming message alerts that a new market emerges. It is desirable to get more specific information by sending surveys to those targeted customers. However, this kind of behaviour cannot be predicted beforehand: the survey content can only be constructed with the type of market known. How can new instructions be added to cope with this emerging opportunity?

Popular solutions: For most systems, it is not possible to alter system behaviour during runtime. System rebuild is required to alter system behaviour.

Event Web solution: How an entity reacts toward events is controlled by a set of Event Chains. It is specified in XML format to ensure platform independence, and it can be easily converted into events and sent among nodes. Processor, which acts like an interpreter, is introduced to execute the Event Chain upon event arrival. The Event Chain is stored as a DOM tree in the Processor which can be replaced during runtime when system change is needed.

The Event Chains adapted in the implementation consist of state transition diagrams and event-driven functions:

- ✓ The state transition diagrams are used to model the system behaviour as a state machine, *e.g.*, if the incoming event indicates that more than 100 people are interested in this product, the state becomes "hot". State transition diagrams can also represent how the order of event arrival affects the result.
- ✓ The event-driven functions are used to fully model the event-based collaboration model, *e.g.*, "terminate" events always halt the system, regardless of the current state.

Users can also specify the responses towards uncorrelated events in different Event Chains. This is allowed as the Event Server is designed in a way that an entity can be manipulated by multiple Event Chains. Each Chain can be changed without affecting the others during runtime.

Lastly, control structures, objects and functions are available for specifying the Event Chain. This is to add representational advantage of programming language into Event Chains. [2] Simple Event Chains can be represented by state diagrams, which can be constructed by visual tools and can be handled by

non-programmers. For experienced programmer, the full power of Event Chains can be revealed by applying language pattern to the design of Event Chain.

In this example, after the Event Server is notified about this new opportunity, an Event Chain can be added dynamically to construct the specific surveys which are then sent to customers. Based on customers' responses, more specific actions can be made by modifying the Event Chains.

5.2.2. Rapid Development

Scenario: A new department is added to cope with an emerging market. The server of this new department has to connect to all the other departments in the enterprise, which are distributed in different geographical locations. How can this be done with minimal disturbance?

Popular solutions: As shown previously, connecting more nodes requires system rebuild in Web Services and remoting. The changes have to be made synchronously. In addition, there is no way to modify system behaviour during runtime. As a result, considerable latency and resources are needed to complete this action.

Event Web solution: Each entity has a set of Agents, which encapsulate the state information. Upon event arrival, Event Chains will be invoked to perform state transformation on the Agents, based on the rules in the Event Chain and the content of the incoming event. After that, an outgoing event will be generated to notify other entities about its changes.

In this example, the new department has Agents "Customers" and "Resources". Agent "Customers" contains variables representing the information of customers: their favourite fashions, shopping habits, *etc.* These variables represent the current market status observed. "Customers" will be updated by incoming events, which are received from event source "customer survey centre" as it has previously subscribed. Upon event arrival, the Event Chain will be executed to update the state information, *e.g.*, "if more than 100 customers are interested in this fashion, mark this fashion as 'hot'".

This model ensures an entity can grab and perform analysis on the hottest news within the shortest time:

- ✓ Due to the dynamic nature and flexibility of the event subscription mechanism, it is easy to navigate to the channel which gives the newest information.
- ✓ Due to the many-to-many and asynchronous nature of the event-driven communication model, parallel information processing is possible, thus minimizing the latency in analyzing the hottest news.
- ✓ How the event updates the Agent is controlled by Event Chains. As mentioned, Event Chains can be changed dynamically, so it is easy to form new strategy to cope with new situations during runtime.

As a result, Agents representing this newly formed department are added and connected to others. In order to minimize the disturbance, a mechanism is introduced to add Agents during runtime without changing other existing Agents. When an Agent is added, users can specify the nature of the Agent by declaring variables, *e.g.*, customers' shopping habit, income group, *etc.* To receive the appropriate events, Agent subscribes events that are of its interests, *e.g.*, events from "customer survey centre". Event Chains can be added on-the-fly to instruct how Agents respond to those events. As a summary, rapid development can be done as follows:

- ✓ Agents are added during runtime to represent the presence of new entities.
- ✓ Event subscription is changed during runtime to navigate for the newest information source.
- ✓ Event Chains are added on-the-fly to give the newest instruction for data analysis.

5.3. Variations in Communications

5.3.1. Connecting nodes of different devices and platforms

Scenario: In order to deliver the newest product information, salesmen are now equipped with mobile devices to get updated with the server at anywhere and at anytime. These mobile devices may be laptops, PDAs or smartphones. Interaction between the server and the salesmen is necessary to find out the specific situations so as to deliver the most precise information. How to enable such interaction when the mobile devices are of different hardware and software specifications?

Current solutions: In the very beginning, only SMS or e-mails can be consumed by mobile devices universally. As a result, the interaction between servers and clients can only depend on text-based messages, which is very inefficient: someone must sit at the server side to cope with these responses manually.

In the case of remoting, servers and clients communicate with remote procedure calls, thus interaction is possible. However, since they are communicating in binary format, the entities have to agree on the language and platform used. As a result, remoting fails in connecting nodes of different software specifications.

In Web Services, nodes with different platforms can be connected together. The service contracts are described in XML, which is universal for many platforms. In addition, the end users are able to consume more complex messages like datasets, which helps them gain accessibility to the database of the server.

However, all these are still inadequate in this example. In order to find out the specific situations that each salesman is in so as to deliver the most precise data, the server may want to send a survey for the salesmen to fill in. To make this happen, the mobile clients should be able to consume services that construct user interfaces on the clients, with the appearance of the survey encapsulated in the message. However, the hardware specifications (*e.g.*, screen size and input tool) are different for different devices. Thus the server cannot describe the appearance of the survey precisely for different devices unless the server learns about all these different specifications, which is impossible. As a result, for the case of consuming a survey, the user interfaces must be fixed in the devices beforehand and only consumes the survey questions in text-based format. This gives two disadvantages:

- ✓ User interfaces have to be re-written for different devices, which waste resources.
- ✓ The format of the survey has to be fixed beforehand, which gives inflexibilities and is against the ‘sense-and-respond’ idea.

Event Web solution: First, interoperability is ensured as the messages, which are events, are in XML format: it has broad user support and XML parsers exist for most platforms. Second, in order to consume messages of complex types optimally on most devices, different types of Actuators are introduced in the Mobile Event Processor. When an event arrives, the Actuator of that particular type will be invoked to process the event.

Figure 8: The same message encapsulating details of a survey can be sent to all mobile devices without learning the hardware and software specifications of all mobile devices.

In this example, an event that constructs a user interface representing the survey is sent to the Mobile Event Processor that is installed on different types of mobile devices, asking for specific details of customers. An event describing the interaction with the user interface will be sent back automatically. Based on the specific details of the customers, the Event Server can choose the best product to deliver, which can be sent as a dataset or represented by a user interface, if more complex interaction is again needed. In addition, if some products in the dataset of the Mobile Event Processor get outdated, the Event Server can send instructions to it directly and delete those entries.

5.3.2. Change Roles

Scenario: The server wants to push out a service recorder for its mobile clients, so as to monitor the salesmen equipped with mobile devices, who work in different parts of the world. How can this be done?

Popular solutions: This is not possible for any popular solutions as the role of servers and clients is very static: it is not possible for a 'server' to become a 'client' and initiates the connection to request for services. However, this kind of model is inadequate as mobile devices emerge and are becoming more powerful. In this case, the servers have to require services so as to get on-site information from the mobile clients. There are four constraints here:

- ✓ If the server wants to initiate connections, it must learn about all specifications of clients before it can send the optimal request, which is very inefficient.
- ✓ The clients may not be capable of completing the heavy computational job to provide the service.
- ✓ The environment of different clients may or may not be well set to provide the service, thus timing constraint exists in requesting the service.
- ✓ A mechanism may be needed for the clients to keep state for the servers so that future request can take reference to previous ones.

Event Web solution: The fours constraints can be overcome as follows:

- ✓ The first constraint is especially true for mobile clients, where hardware and software specifications

are mostly different. The solution to this problem is shown previously, where different Actuators are made to consume events optimally on different devices.

- ✓ The second problem is about the evaluation power of mobile clients, as the clients may not be able to provide services that are too heavily-loaded. The solution is shown previously, where the appropriate Web Services can be chosen during runtime to handle the heavily loaded task.
- ✓ The third problem is about fulfilling the pre-requisites of providing services. This kind of exception is more common in server-initiated connections since servers are centralized and maintained by system engineers, while clients are distributed and just maintained by end users. Thus a more efficient way to handle these exceptions should be made. The solution to this problem is to make use of the event buffering mechanism and the ability to modify clients' behaviour on-the-fly. For example, a mobile client may be requested to hand in records that have not been collected (maybe due to the laziness of the salesman). The mobile client can buffer this request and report this situation to the server. When the server recognizes this, the server may delete all database records in the clients and indicate the worker is fired due to his laziness. As a result, exceptions can be coped with minimal latency as the server can interfere with the behaviour of remote clients explicitly.
- ✓ The fourth problem is about bookkeeping of state at the clients' side. It may be advantageous to keep states at the clients such that future requests to server can be based on previous ones. Thus, Agent is reused in this case to keep states of the clients, as well as to represent the clients' responses towards events. This gives another advantage that the server does not need to keep states for the clients, which reduces the load of the server and increases scalability. In addition, this introduces loose-coupling between servers and clients. Clients can shift to server anytime since it is keeping its own states, thus the new server can learn about the state history directly from the clients. However, this state keeping mechanism should not be casually used as too much data may crash the mobile devices with limited memory.

5.4. **Programming Tasks**

5.4.1. Perform Heavy Computational Tasks on Mobile Devices

Scenario: A programmer is alerted that a message of unknown type has reached the server. He wants to add instructions to the server to cope with this unknown message. However, he is away from the server and only has a mobile device in hand. What can he do in order to change the system in the shortest time?

Popular Solutions: In this example, the best way in connecting mobile devices to main server is by Web Services, due to its interoperable nature. The programmer can change the system by consuming the Web Services. This implies the instruction made to the system must be predicted beforehand, since the service cannot be changed dynamically. However, this change is made due to an unknown message, which is not always predictable. Upon unpredictable changes, the programmer can only contact the on-site programmers to add instructions, which is inefficient. Two constraints are shown here: first is the ability to add instructions to systems on-the-fly; second is the ability for mobile devices to generate instructions from users input, which may require heavy computations. Complex computing is inefficient on mobile devices due to its limited computational power and battery life.

Event Web solution: The first constraint is discussed previously. For the second constraint, two milestones have to be reached in order to overcome it. First is the ease for the programmer to describe instructions in mobile devices with limited hardware capabilities. Special visual tools are needed to construct the graphical representation of the instructions and will be discussed in the next example. The other is to overcome the low computational power and battery life of mobile devices, as converting graphical representation of instructions to real ones may need complex computing. Upon this, Web Services can be consumed to complete the job. Such Web Services servers are likely to be maintained by the same enterprise so as to enhance the overall performance of the entire system. Thus using Web Services is feasible in this case as the service contracts and location of servers are mostly constant.

However, the type of Web Services consumed can only be known upon runtime: it entirely depends on the requests from users or incoming events. This brings difficulties as in Web Services, the entities are connected by static service binding. The approach adapted is to make a gateway for consuming different Web Services. The Web Services Actuator calls the same Web Services every time whenever Web Services should be called. The specific request details will be passed to this "Gateway Web Services". This "Gateway Web Services" will then multiplex and forward the request to the right Web Services.

Tasks requiring constant algorithm changes can be delegated to Web Services as well. When the algorithm of the Web Services is changed, no change is needed on the clients as long as the service contracts remain constant. However, if that module is embedded in the distributed software, redistribution of the software is needed whenever there is a change.

5.4.2. User Interfaces

Scenario: Is it possible to have a visual tool representing the system so that even non-programmers can change the system behaviour?

Popular Solutions: No such visual tool is provided for connecting nodes and specifying interactions between the nodes in any currently available solutions.

Event Web Solution: The components that characterize the connection between nodes can be easily represented by pictures. For example, Agents can be represented by tables with columns "variable name" and "variable value"; Event Chains can be represented by state diagrams; the subscription directory can be represented by a table with columns "interest" and "destination addresses".

5.5. Summary

As a summary, Event Web achieves the followings when compared to other popular solutions like Web Services or remoting:

- ✓ More flexibility and dynamics in:
  - ➢ Connecting different nodes in network.
  - ➢ Changing the nature and interest of an entity represented by Agents and event subscriptions.
  - ➢ Changing roles of different nodes in network.
  - ➢ Changing responses towards events represented in Event Chains.
- ✓ More powerful in modelling the dynamic environment:
  - ➢ State concepts illustrated in Agents.

- Expressiveness brought by Event Chains to model responses towards events.

✓ More varieties in communication:
- Many-to-many instead of one-to-one, enabling parallel processing of information.
- Able to connect nodes with different software or hardware specifications.

✓ Easier for non-programmers to understand:
- Provide visual tools to model the system.
- Programming concepts are encapsulated.

✓ Better utilization of mobile devices:
- Get on-site information expressed in a powerful manner by events.
- Ability to consume complex events optimally by Actuators.
- A custom communication scheme.

An adaptive enterprise can make use of these special capabilities to radar for new markets by sensing and analyzing a wide scope of information with the shortest latency. Making topology or system changes become easy, seamless and can be done by non-technical staff. In order to have the best performance, the system is implemented with care. For example, in order to work well with mobile devices, we have developed a scheme to cope with hands off. We have developed security scheme to authenticate users, who changes their locations often if they are the mobile nodes. In order to ease re-development, design patterns [9] are extensively used in system design.

6. Conclusion

In this thesis, the implementations of the Event Server and the Mobile Event Processor are demonstrated. The usefulness of the Event Web on solving the problems faced by an adaptive enterprise is discussed. Comparisons with other popular solutions like Web Services and remoting shows that the Event Web is particularly suitable for enterprises which need to adapt to fast-changing environments.

7. References

[1] Stephan H. Haeckel. Adaptive Enterprise. Harvard Business School Press, c1999.

[2] Kenneth C. Louden. Programming Languages, Principles and Practice. Thomson Learning, 2003.

[3] Service Deployment Pattern, Microsoft Press.

[4] K. Mani Chandy, Brian Emre Aydemir, Elliott Michael Karpilovsky, and Daniel M. Zimmerman. *Event-driven architectures for distributed crisis management*. 15th IASTED International Conference on Parallel and Distributed Computing and Systems, 2003.

[5] MSDN Online Library: http://msdn.microsoft.com/library/default.asp

[6] .NET XML Web Services, Microsoft Press

[7] .NET ADO, Microsoft Press

[8] Y Natis. *Events Will Transform Application Servers.* Gartner Research, 2003

[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series.

8. Appendix

8.1. Message Format / XML File Format

8.1.1. Message for Requesting Creation of New Agent Type

```
<root>
<Type>agentRequest</Type>
<Scope>new</Scope>
<agentType>UI</agentType>
</root>
```

8.1.2. Message for Creating New Agent Type

```
<root>
<Type>agentDef</Type>
<Scope>local</Scope>
<agentDef>
        <agentType>UI</agentType>
        <actuatorName>UIProto</actuatorName>
        <info><!-- list of name-value pairs for state variables -->
                <variable>
                        <name>sample</name>
                        <value>hello</value>
                </variable>
        </info>
</agentDef>
</root>
```

8.1.3. Message Specifying User Interfaces

```
<root>
  <Type>UI</Type>
  <Scope>test</Scope>
  <UI>
    <Template>SurveyControls</Template>
    <Controls>
      <ID>1</ID>
      <Type>MultipleChoiceQ</Type>
      <Name>q1</Name>
      <Parameters>
        <Question>Choose one of the following</Question>
        <Choice>1|2|3</Choice>
        <DefaultAns>2</DefaultAns>
        <Number>3</Number>
      </Parameters>
      <ControlInfo>
        <!--detail not fixed yet-->
      </ControlInfo>
      <Event>
      </Event>
    </Controls>
    <Controls>
      <ID>2</ID>
      <Type>OpenEndQ</Type>
      <Name>q2</Name>
      <Parameters>
        <Question>Enter something</Question>
        <DefaultAns>
        </DefaultAns>
        <Number>2</Number>
      </Parameters>
      <ControlInfo>
        <!--detail not fixed yet-->
      </ControlInfo>
      <Event>
      </Event>
    </Controls>
    <End>
      <!--Event to be performed after the all the UI is manipulated etc., not fixed yet-->
    </End>
  </UI>
</root>
```

8.1.4. Message Storing User Input

```
<root>
<Type>Result</Type>
<Scope>Result</Scope>
<Result><!-- list of user responses -->
      <q1>1</q1>
      <q2>hahaha|hihihi|</q2>
</Result>
</root>
```

8.1.5. Message Specifying State Transformation

```
<root>
<Type>UI</Type>
<Scope>scope1</Scope>
<actuatorName>transformer</actuatorName><!-- override the default actuator used by the agent -->
<transform>
	<condition>string eq haha|</condition><!-- condition to trigger state transformation; “Nil” for unconditional transform -->
	<detail>
		<change>
			<transformType>insert</transformType><!-- insert a new state variable -->
			<name>satisfied</name>
			<method>
				<add>yes</add><!-- method to generate the value for the variable, either “add” or “operate” -->
			</method>
		</change>
		<change>
			<transformType>delete</transformType><!-- delete this state variable -->
			<name>sample</name>
		</change>
		<change>
			<transformType>update</transformType><!-- update this state variable -->
			<name>string</name>
			<method>
				<operate>string + haha</operate><!-- method to generate the value for the variable, either “replace” or “operate” -->
			</method>
		</change>
	</detail>
</transform>
</root>
```

8.1.6. Message for Storing State Information

```
<root>
<Type>UI</Type>
<Scope>scope1</Scope>
<state><!-- list of state variables -->
	<type>UI</type>
	<sample>hello</sample>
	<scope>scope1</scope>
	<string>haha|</string>
	<number>2</number>
</state>
</root>
```

8.1.7. Message to Create a New Agent to Handle Database Object

```
<root>
<Type>agentDef</Type>
<Scope>local</Scope>
<agentDef>
        <agentType>data</agentType>
        <actuatorName>data</actuatorName>
        <info>
        </info>
</agentDef>
</root>
```

8.1.8. Database Event

```
<root>
<Type>data</Type> <!-- fix-->
<Scope>data1</Scope> <!-- scope name-->
<Data> <!-- fix -->
<dataset> <!-- dataset serialized code -->
<course>
<code>410</code>
<title>wireless</title>
<credit>3</credit>
</course>
<course>
<code>3821</code>
<title>lab</title>
<credit>2</credit>
</course>
</dataset>
</Data>
</root>
```

8.1.9. An Event Chain for Requesting New Instructions for State Transformation

```
<Root>
<State>
<Statename>start</Statename> <!--map to the current state and extract the rules under the node-->
<Action actuator="send">
        <Return>true</Return>
        <Change>true</Change>
        <Property>agentVar|ID|11|Choice</Property>
        <Property>state1|ID|1|DefaultAns</Property>
        <File>surveyChain.xml</File>
        <Target>mpchau2</Target>
</Action> <!--action to be perform under this state-->
<Newstate>wait1</Newstate> <!--change to new state-->
</State>
<State>
<Statename>wait1</Statename>
<If>
    <Predicate source="eventtype">Type eq reply2</Predicate>
    <True> <!--execute these rule-->
    <Action actuator="send">
        <Return>true</Return>
        <Change>false</Change>
        <Property></Property>
        <File>create.xml</File>
        <Target>mpchau2</Target>
    </Action>
    <Newstate>wait2</Newstate>
    </True>
</If>
<Else if="true">
<Predicate source="eventtype">Type eq agentRequest</Predicate> <!--otherwise if the event is 'agentRequest'-->
<True>
<!--execute these rules -->
    <Action actuator="send">
        <Return>false</Return>
        <Change>false</Change>
        <Property></Property>
        <File>agentDefUI.xml</File>
        <Target>mpchau2</Target>
    </Action>
    <Newstate>wait1</Newstate>
</True>
</Else>
</State>

<State>
<Statename>wait2</Statename>
<Action actuator="send">
    <Return>true</Return>
    <Change>true</Change>
    <Property>agentVar|ID|11|Choice</Property>
    <Property>state2|ID|1|DefaultAns</Property>
    <File>surveyChain.xml</File>
    <Target>mpchau2</Target>
</Action> <!--action to be perform under this state-->
<Newstate>wait3</Newstate> <!--change to new state-->
</State>
```

```
<State>
<Statename>wait3</Statename>
        <Action actuator="send">
                <Return>true</Return>
                <Change>false</Change>
                <Property></Property>
                <File>create.xml</File>
                <Target>mpchau2</Target>
        </Action>
        <Newstate>wait4</Newstate>
</State>

<State>
<Statename>wait4</Statename>
<Newstate>end</Newstate>
</State>
<State>
<Statename>end</Statename> <!-- terminating state -->
</State>
...
<Global>
<Timeout> <!--if the incoming event is timeout, execute the following rules, no matter what state it is-->
        <Action actuator="system">sleep 10 second</Action>
</Timeout>
</Global>
</Root>
```

8.1.10. A Fragment of Survey Event which Requests for Event Chain Details

```
<root>
   <Type>UI</Type>
   <Scope>create</Scope>
   <ReturnType>reply2</ReturnType>
   <ReturnScope>wet</ReturnScope>
   <UI>
      <Template>SurveyControls</Template>
      <Controls>
         <ID>1</ID>
         <Type>OpenEndQ</Type>
         <Name>Statename</Name>
         <Parameters>
            <Question>State name:</Question>
            <DefaultAns>state2</DefaultAns>
            <Enable>false</Enable>
            <Number>1</Number>
         </Parameters>
         <ControlInfo>
            <!--detail not fixed yet-->
         </ControlInfo>
         <Event>
         </Event>
      </Controls>
      <Controls>
         <ID>2</ID>
         <Type>OpenEndQ</Type>
         <Name>Predicate</Name>
         <Parameters>
<Question>Conditioning?</Question>
            <DefaultAns>ha eq u</DefaultAns>
            <Number>1</Number>
         </Parameters>
         <ControlInfo>
            <!--detail not fixed yet-->
         </ControlInfo>
         <Event>
         </Event>
      </Controls>
      <Controls>
         <ID>3</ID>
         <Type>MultipleChoiceQ</Type>
         <Name>Source</Name>
         <Parameters>
            <Question>Source of
predicate?</Question>
            <Choice>event|agent</Choice>
            <DefaultAns>event</DefaultAns>
            <Number>2</Number>
         </Parameters>
         <ControlInfo>
            <!--detail not fixed yet-->
         </ControlInfo>
         <Event>
         </Event>
      </Controls>
<Controls>
         <ID>4</ID>
         <Type>MultipleChoiceQ</Type>
         <Name>ActionTrue</Name>
         <Parameters>
            <Question>Condition for action?</Question>
            <Choice>none|true|false</Choice>
            <DefaultAns>true</DefaultAns>
            <Number>3</Number>
         </Parameters>
         <ControlInfo>
            <!--detail not fixed yet-->
         </ControlInfo>
         <Event>
         </Event>
      </Controls>
      <Controls>
         <ID>5</ID>
         <Type>OpenEndQ</Type>
         <Name>File</Name>
         <Parameters>
            <Question>(Action) What is the file
name?</Question>
            <DefaultAns>vote.xml</DefaultAns>
            <Number>1</Number>
         </Parameters>
         <ControlInfo>
            <!--detail not fixed yet-->
         </ControlInfo>
         <Event>
         </Event>
      </Controls>
      <Controls>
         <ID>6</ID>
         <Type>MultipleChoiceQ</Type>
         <Name>Target</Name>
         <Parameters>
            <Question>(Action) target
sender?</Question>
            <Choice>khhui2|mpchau2</Choice>
            <DefaultAns>mpchau2</DefaultAns>
            <Number>2</Number>
         </Parameters>
         <ControlInfo>
            <!--detail not fixed yet-->
         </ControlInfo>
         <Event>
         </Event>
      </Controls>
```

8.1.11. An Event Chain Generated as a Result of Consuming a Survey Event

```
<Root>
<State>
<Statename>state1</Statename>
<If>
	<Predicate source="event">ha eq u</Predicate>
		<True>
			<Action actuator="send">
				<File>vote.xml</File>
				<Target>mpchau2</Target>
				<Return>no</Return>
				<Change>no</Change>
				<Property>(null)</Property>
			</Action>
		</True>
	</If>
	<Else if="false">
		<Update>
			<name>abc</name>
			<method>
				<replace>HA</replace>
			</method>
		</Update>
	</Else>
	<Newstate>state2</Newstate>
</State>
<State>
<Statename>state2</Statename>
<If>
	<Predicate source="event">abc eq (null)</Predicate>
		<True />
</If>
<Else if="false">
	<Action actuator="send">
		<File>reason.xml</File>
		<Target>mpchau2</Target>
		<Return>no</Return>
		<Change>yes</Change>
		<Property>agentVar|ID|9|Choice</Property>
	</Action>
	<Update>
		<name>e</name>
		<method>
			<replace>(null)</replace>
		</method>
	</Update>
</Else>
<Newstate>state3</Newstate>
</State>
</Root>
```