VIRTUAL INTERNET PETS BASED ON JAVA-ENABLED MOBILE AGENTS

Gautam Gupta, Samridhi Ganeriwalla, Sanju Sunny, Stuti Nautiyal
Centre for Advanced Information Systems
Nanyang Technological University
Singapore –637658

Abstract - *A novel Internet e-business application, titled Virtual Internet Pets (or VIPs for short), which provides a blend of both entertainment and education, and a human-computer interface to simulate life-like pets and interactions with them is presented in this paper. The virtual pets incorporate distinguished features like mobility, autonomy, location tracking and flexible routing with the help of Java-based Mobile Agents. This technical report focuses on the main features of the application and the system architecture to support them.*

I. INTRODUCTION

A. Main Features of VIPs

The main features of the Virtual Internet Pets application that make it useful and unique are:

- **Mobility:** The virtual pets can carry their code and data state with them from one computer to another across the network.
- **Interactions:** The virtual pets can interact with local entities, such as other pets, databases, and file servers which allows them to achieve useful functions.
- **Autonomy:** Algorithms implemented in the code enable the pets to make local decisions on what to do, where to go and when to go.
- **Flexible Itinerary:** The pet need not travel in predetermined routes; it can also modify its own route dynamically as it acquires additional information during its journey.
- **Location Tracking:** This feature allows the owner to track down the whereabouts and status of their pets.
- **Realistic 3D Animations:** The pets can be seen in action as if they are real and alive**.**

Mobile agents

As the virtual pets may move from host to host in a flexible fashion, a conventional client-server framework does not seem to fit the application well. As it turns out, all the features mentioned in the preceding paragraph can be supported seamlessly by using the Java-based mobile agents [1,3,4]. For instance, a virtual pet impersonated by a mobile agent can communicate with both local and remote pets (agents) and is capable of migrating to remote hosts selectively.

Besides mobility, the application also needs to simulate pets that are autonomous, have feelings and emotions, and are able to interact with their counterparts. This indeed poses a great challenge, and meeting the challenge by using Java would demonstrate that Java is indeed mature and general-purpose for Internet applications.

B. Synopsis of Report

This paper presents the architecture, design and implementation issues and the technologies deployed to create the virtual pet. Because of the analogy between agents and pets, the two terms are often intermixed in the remainder of the report. Section III describes the system architecture, explaining all the components of the application: the Mobile Agent Monitor and the Graphical User Interface. Section IV explains how the key features of our application: mobility, autonomy, interaction, flexible routing, location tracking have been effectively achieved using Java and the mobile agent paradigm

II. SYSTEM ARCHITECTURE

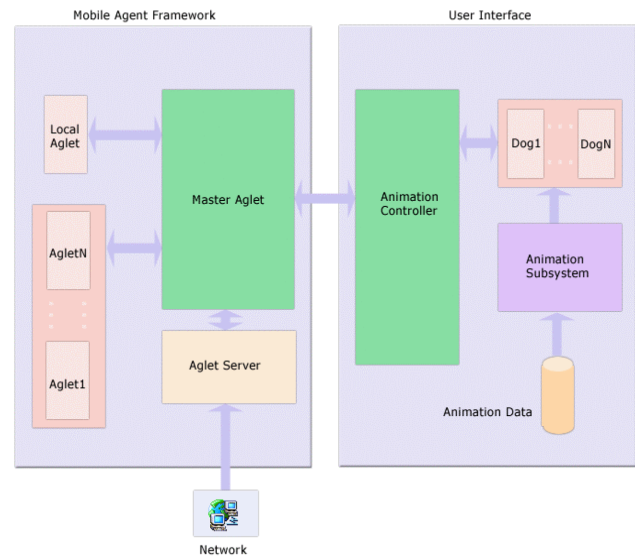

Fig. 1 System Architecture

Fig. 1 shows the architecture of the system. Notably, two core components interact together to achieve the functionality of the application, namely:

- Mobile Agent Monitor
- Graphical User Interface

A. Mobile Agent Monitor

A mobile agent can transport itself from one system in a network to another. The ability to travel allows mobile agents to move to a system that contains objects with which they want to interact and then take advantage of being in the same host or network as the object. When a mobile-agent moves, it brings along with it, its program code as well as the data state (content of the instance variables).

1) ***Agent Server:*** Each machine that intends to host mobile agents must provide a protected agent-execution environment. Such *agent servers* execute agent code and support primitive operations: allow agents to migrate from one host to another, create an execution environment and grant controlled access to its local resources [2].

The VIPs application embeds and extends the Agent Server to provide a greater level of functionality as required by the system. A generic standalone Agents server was deployed, which would be extensible in the future to suit specific requirements. The server allows the monitoring of the activities of the agents in the context and responds to these activities.

An *agent context* is a stationary object that provides a means of maintaining and managing running agents in a uniform execution environment. The agent context is created by a server that has a network daemon whose job is to monitor the network for agents. Incoming agents are received and inserted into the context by the daemon. It also contains a security manager that protects the underlying host. This server keeps an agent listener, monitoring all the actions made by the agents (creation, moves, disposal)[1].

2) ***Stationary Master Agent****:* The Agent Server is responsible for creating the Stationary Master Agent whose basic functionality is to:

- interact with the user via the user interface.
- create the local agent, dispatch agents to do "work" at remote hosts and retract the agent back if necessary.
- manage all user interface-agent and agent-agent communication.

The Stationary Master Agent is the center of control. It manages messages from the user interface and forwards them to the required agents using peer-to-peer asynchronous message passing. Agents residing in the current context may reply to the message, which is routed to the Graphical User Interface to update the display. One-way message passing is used to avoid deadlocks and the situation where the message handler goes into a non-terminating loop.

The Stationary Agent creates the local agent and hence acquires a proxy to communicate and gain access to the corresponding agent. A *proxy* acts as a shield that protects an agent from malicious host and also provides location transparency. The Master Agent provides the essential parameters (its own ID) to the agent on its creation, to facilitate future communication.

To move a pet, the following operations may apply to the corresponding agent: 1). *Dispatching*, i.e., request it to go to another location to continue execution there; 2) *Retracting, i.e.,* to request it to return from the remote location. The request to the local agent is made by the Stationary Master Agent in response to a user-initiated action.

Multiple agents originate from different sources and meet in a specific context. *Message Multicasting* allows a single agent to post a message to a group of agents (subscribers)[1]. The Stationary Master Agent subscribes to the particular message sent out by incoming agents to advertise their arrival, and also implements the handler for it. The message provides the Stationary Master Agent with parameters required for future communication. The Stationary Master Agent also informs the local agent of the arrival. After which, message passing between the agents residing in the same context is via the Master Agent. When the agent is leaving the host, because of retraction or after completion of its task, it explicitly informs the Master Agent, which then requests the user interface to remove the agent from the current display.

The framework also supports *remote messaging* for communications between pets that reside on different hosts. It does not cause any transfer of byte code. Stationary Master Agents residing on different hosts use remote messaging to transfer information between users. The local agent when dispatched to a remote location, can use remote messaging to inform its corresponding Master Agent about its status.

3) ***Local/Incoming Agents:*** The local agent (hence, pet) is created by the Stationary Master Agent. It is inserted into the current context and assigned a separate thread of execution. Each agent is assigned a globally unique identifier, and immutable throughout the lifetime of the agent [1].

Constrained randomness has been introduced using various counters that represent the state of the agents. Agent events cause the counter values to change. A separate thread is used to monitor the counters, such that if it exceeds a certain threshold, the agent triggers certain events, which create the uncertainty.

The agent can navigate independently to multiple hosts based on its itinerary specified by the user. It might even modify its itinerary dynamically depending on local information. The Master Agent initiates dispatching of the local agent. On request, the code and data of the agent is

serialized, removed from its context, all its threads of execution killed and finally it is transferred to its destination.

On arrival, the incoming agent multicasts a message advertising its arrival, which is handled by the Master Agent. The Master Agent messages back with appropriate parameters, which are required for future communication.

To obtain status information of the roaming agent and to enable retraction by the Master Agent, location tracking is incorporated. When the agent moves from one location to another, it leaves behind an electronic fingerprint, indicating the completion of the task at that location. On user's request, the Master Agent can send a search agent, which is provided with the same itinerary as the mobile agent. The search agent checks for the fingerprint at each destination and then determines whether the agent is present at that destination currently. If so, it gets the agent's proxy and returns to the Master Agent with the information. The Master Agent can then retract its local agent. This feature of location tracking avoids losing track of any pet at any time, and gives the home location the authority to retrieve the agent.

B. The User Interface

The user interface is a critical part of any application and particularly so in this case. It is embedded in the Master Agent and can receive messages from both the Master Agent and the user. The Master Agent acts as a communication link between the resident Agents and their on-screen entities. Key components and features of the interface are outlined below.

1) ***Animation Controller****:* The animation controller is responsible for handling the overall display components. These include dynamic backgrounds, character animations and interface components like pop-up menus. Swing components (Java Foundation Classes) are used as they support Pluggable Look and Feel and more importantly, double-buffering. Pluggable Look and Feel ensures that the user interface components visually reflect the corresponding components of the underlying operating system. Double buffering is used to eliminate 'flickers'. The animation controller has an animation loop that prepares each frame by drawing the graphics in the correct order. A timer thread running in the background repeatedly calls this function and updates the display.

2) ***Characters****:* A character is created when a new agent is added to the current context. Each such object is responsible for handling and updating its display. Characters also include non-interactive members.

3) ***Realistic 3D Animation****:* Animations are pre-rendered in 3D against a blue background. The 3D model of the pet is manipulated and key-framing is used to generate an animation clip. The individual frames are then post-processed to add transparency and exported to the gif89a format (which supports transparency). The frames are then displayed one after another to create the illusion of motion.

4) ***Intelligent Interaction****:* The most critical and interesting part of the interface is the way it handles intelligent interaction. This is achieved through position tracking and an animation queue. Autonomous events generated by an agent or user interface events are processed by the Master Agent and a unique state change code is sent to the interface. The controller routes this message to the correct agent on-screen entity. This entity with the help of the animation data inserts the correct sequence of animation clips required to generate the logical state change. Animations in the queue are executed in sequence until another code arrives or the queue becomes empty. When the queue is empty (Agent is idle), a new animation is inserted based on the current animation and a random factor.

III. DISCUSSIONS AND CONCLUSION

In summary, the main features of the virtual Internet pets application are supported as follows:

- **Mobility:** The virtual pets can carry their code and data state with them from one computer to another across the network. The platform independence nature of Java allows us to create mobile agents without knowing the type of computers it is going to run on. The security architecture of Java makes it reasonably safe to host an un-trusted agent, because it cannot tamper with the host or access private information. The virtual machine loads and defines classes at runtime (dynamic class loading). The class-loading mechanism is extensible and enables classes to be loaded via the network [1]. The key feature that supports mobility is that the agents can be serialized and de-serialized.
- **Autonomy:** Algorithms implemented in the code of agents enable the pets to make local decisions on what to do, where to go and when to go. Allowing each agent to execute in its own lightweight process, also called a *thread of execution*, is a way of enabling agents to behave independently of others residing within the same context. Java not only allows multithreaded programming but also supports a set of synchronization primitives that enable concurrent actions of the agents[1]. Mobile agents adapt dynamically i.e. have the ability to sense their execution environment and react autonomously to changes.

- **Interaction:** The virtual pets can interact with local entities, such as other pets, databases, and file servers which allows them to achieve useful functions. Agent messaging can be peer-to-peer or broadcasting. For agent communication, one-way-type messaging has been used which is asynchronous and allows the two agents to engage in a loosely coupled conversation in which the message-sending agent does not expect any replies from the message-receiving agent.
- **Flexible Itinerary:** The route traversed by a pet can be predetermined, but it can also be modified dynamically by the agent as it discovers additional information during its journey. Dynamic routing adds flexibility and robustness to the application. The autonomous nature of the agents makes this an easy task.
- **Location Tracking:** This feature allows the owner to track down the whereabouts and status of their pets. A logging approach was devised such that when an agent leaves a host it reports its presence. A search agent sent with the same itinerary will check for this fingerprint on the current host and whether the agent is currently present there. If so, it will return with the status information of the lost agent.

In conclusion, we have been successful in simulating a real pet with the help of the Java based mobile agent framework. This life-like virtual pet would be autonomous, have emotions and also have the ability to migrate to other computers on the Internet to interact with its counterparts. We have also successfully achieved Realism in the animations to great extent.

The scope of expansion for this particular application in the future is tremendous. For one, it would be much desirable to incorporate Artificial Intelligence to make the pets even more interesting. Another dimension for exploitation would be to port the application onto a hand held device. The increased portability would have a tremendous immediate impact.

IV. REFERENCES

[1] Danny B. Lange and Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets: Addison-Wesley Longman, Inc. 1998.

[2] Neeran M. Karnik and Anand R. Tripathi, “Design Issues in Mobile-Agent Programming Systems,” IEEE Concurrency, July/September 1998.

[3] Danny B. Lange, “Java Aglets Appliction Programming Interface (J-AAPI) White Paper – Draft 2,” IBM Tokyo Research Laboratory, February 19,1997.

[4] Mitsuru Oshima, Guenter Karjoth and Kouichi Ono, “Aglets Specification 1.1 Draft,” September 8, 1998.

[5] "IBM Aglets Software Development Kit," [Online document], Available HTTP: http://www.trl.ibm.co.jp/aglets/