OpenPING: A Reflective Middleware for the Construction of Adaptive Networked Game Applications

Paul Okanda
Computing Department,
Lancaster University,
Lancaster, LA1 4YR.
+ 44 (0) 1524 593315
okanda@comp.lancs.ac.uk

Gordon Blair
Computing Department,
Lancaster University,
Lancaster, LA1 4YR.
+ 44 (0) 1524 593809
gordon@comp.lancs.ac.uk

ABSTRACT

The emergence of distributed Virtual Reality (VR) applications that run over the Internet has presented networked game application designers with new challenges. In an environment where the public internet streams multimedia data and is constantly under pressure to deliver over widely heterogeneous user-platforms, there has been a growing need that distributed VR applications be aware of and adapt to frequent variations in their context of execution. In this paper, we argue that in contrast to research efforts targeted at improvement of *scalability*, *persistence* and *responsiveness* capabilities, much less attempts have been aimed at addressing the *flexibility*, *maintainability* and *extensibility* requirements in contemporary distributed VR platforms. We propose the use of structural reflection as an approach that not only addresses these requirements but also offers added value in the form of providing a framework for *scalability*, *persistence* and *responsiveness* that is itself *flexible*, *maintainable* and *extensible.* We also present an adaptive middleware platform implementation called OpenPING[1] that supports our proposal in addressing these requirements.

Keywords

Virtual Reality (VR), Middleware Platforms, Reflection, Adaptation, Networked Games.

1. INTRODUCTION

Recent research in networked games and VR service platforms has been aimed at offering adequate support for applications running on the public Internet. This has proved extremely challenging, particularly in massively multi-participant games where thousands of users potentially interact with each other and with thousands of autonomous entities using uncontrolled network and local (processor, memory) resources. In an effort to better address these challenges, researchers have identified various capabilities that a networked games middleware platform should offer. Requirements for such systems include the following:

- Scalability: ability to continue functioning satisfactorily as the system's execution context changes in size or volume in order to meet diverse needs.
- Persistence: capacity to remain active even when some/all user sessions have terminated.
- Responsiveness: capability of responding to user demands within a prescribed time frame guaranteeing sustained support for high levels of interaction between many users.
- Flexibility: ability to satisfy differing system constraints and user needs with fluctuations in the system's execution environment.
- Maintainability: the ease with which the game application can be modified to correct faults, improve performance, or other attributes.
- Extensibility: the ease with which the platform can be altered to increase the system's functional capacity.

The main focus of research on VR and networked game platforms has been on the first three capabilities and, as a result, a number of techniques both at the platform and the application level have emerged:

- To improve scalability, existing published works propose a wide range of world partitioning approaches from static coarse-grained partitions [2] and interest management (perception-based) approaches in Virtual Society [6] to information aggregation depending on level of details.
- To address persistence requirements, some service platforms such as Continuum [4] implement mastership transfer within peer-to-peer architectures. Others have centralised databases that regularly maintain versions of object states.
- To provide support for responsiveness, researchers have attempted to implement fully distributed architectures together with multicast grouping of clients, e.g. DIVE [2].

(A detailed analysis of techniques used in VR platforms can be accessed in [7]).

In contrast, less research has been carried out on addressing the flexibility, maintainability and extensibility requirements of VR

[1] OpenPING is an enhanced version of **P**latform for **I**nteractive **N**etworked **G**ames, the original non-reflective version having been designed by a number of partners in a EU funded project: PING-IST-1999-11488.

SIGCOMM'04 Workshops, Aug. 30 & Sept. 3, 2004, Portland, OR, USA.

and networked game platforms. We aim to address these requirements using a suitable framework that culminates in the realization of an adaptive middleware platform.

This paper is structured as follows. Section 2 presents a background on open implementation. Section 3 then provides an insight into our overall approach and a description of our system's design. Implementation details and an overall architecture are covered in section 4 followed by details of some experiments and their evaluation in section 5. Section 6 then concludes the paper.

2. TOWARDS OPEN IMPLEMENTATION

We believe that contemporary game platform architectures are unable to cope effectively with flexibility, maintainability and extensibility requirements as a result of two main reasons:

1. Firstly, their *black-box* nature inevitably creates a bias in the performance of the resulting implementation since the platform designers have to decide beforehand and make a choice on the implementation, then lock that decision inside a *black-box*. As shown in **Figure I** below, a *black-box* abstraction presents a single interface that exposes functionality but hides implementation.

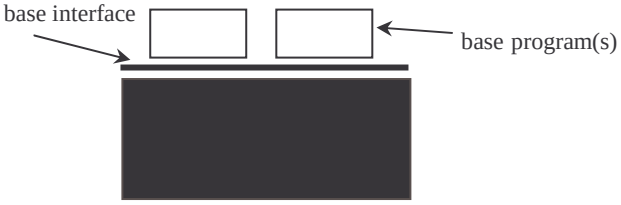

Figure I A *black-box* implementation

2. Secondly, even in instances where access to the platform implementation is enabled, their *highly coupled* nature makes implementation choices of certain services hard-coded in the implementation of others.

Current research in software design is geared towards reconsidering the question of what a system's implementation should expose to its clients. There is huge potential in a system becoming more useful if it allows clients to take control over its implementation strategy. This design principle is called *open implementation* [Kiczales96, DeVolder95].

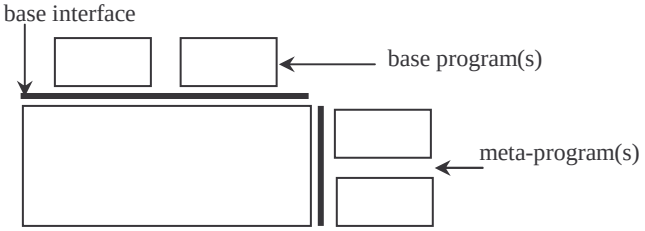

Figure II An *open* implementation

As shown in **Figure II** above, an open implementation presents two interfaces – the primary (base) interface which provides the functionality and the meta-interface which allows the client to adjust the implementation strategy decisions that underlie the primary interface.

Open implementation presents issues such as technologies to support realisation of the above goals and how to make key decisions about what implementation decisions to expose.

We contend that in order to meet flexibility, maintainability and extensibility requirements of networked game middleware, there is an obvious need for more effort towards incorporation of openness in their design.

3. SYSTEM DESIGN

To achieve openness, a primary focus in the research community has been the concept of *computational reflection*. The fundamentals of reflection were introduced by B.C. Smith [11] who argued that a system can be made to control its own representation in much the same way as it controls the representation of its application domain. Such a system is said to have a *self-representation* (also referred to as a meta-representation) whose constituents could be its own state and behaviour. Additionally, if this self-representation has a *causal connection* relationship with the domain, i.e. if a change in the system's self-representation triggers a change in the application domain and conversely, any alteration in the application domain causes a corresponding effect in the system's self-representation, then the system is said to be reflective.

We provide a simple context-specific definition of reflection in networked game applications as;

"a design principle that allows a networked game platform and/or application to have a representation of itself in a manner that makes its adaptation to a changing environment possible".

3.1 Object Model

The framework's design uses an object-oriented approach in which an object consists of:

- a set of accessible attributes,
- a set of methods to get and set these attributes (collectively forming the interface of the object),
- a set of associated behaviours,
- one or more renderings of the object.

Active objects (e.g. avatars) possess all the four elements while passive objects (e.g. components of a networked game terrain) contain all elements except the set of behaviours.

We look into an object model that has three categories of associated behaviours:

- Application (shallow) behaviours: are application level and may or may not trigger changes in the system. For example, the simulation of an avatar's change in location (motion) is an application behaviour.
- Platform (deep) behaviours: are system level and exist at the application level as representations of middleware services or mechanisms. For example, a particular consistency policy that implements a receive-order sequence of events is a platform behaviour.
- Hybrid (shallow-deep) behaviours: these are application-system level with an implementation that causally cuts across the entire game platform. For instance, an event channeling protocol that has application-level input in form of packet loss detection is a hybrid behaviour.

4. PLATFORM IMPLEMENTATION

4.1 Overall Architecture

The platform whose overall architecture is described below offers dynamism in networked game applications via exploitation of application-specific semantics and run-time execution

environment awareness. Crucially, it provides the application designer with access to application objects as well as mechanisms encapsulated in six services within a middleware platform called OpenPING. The services, each with its own set of pluggable mechanisms include: *Concurrency, Replication, Interest Management, Persistence, Consistency* and *Event Channelling*.

The diagram below illustrates a framework for the platform's architectural design.

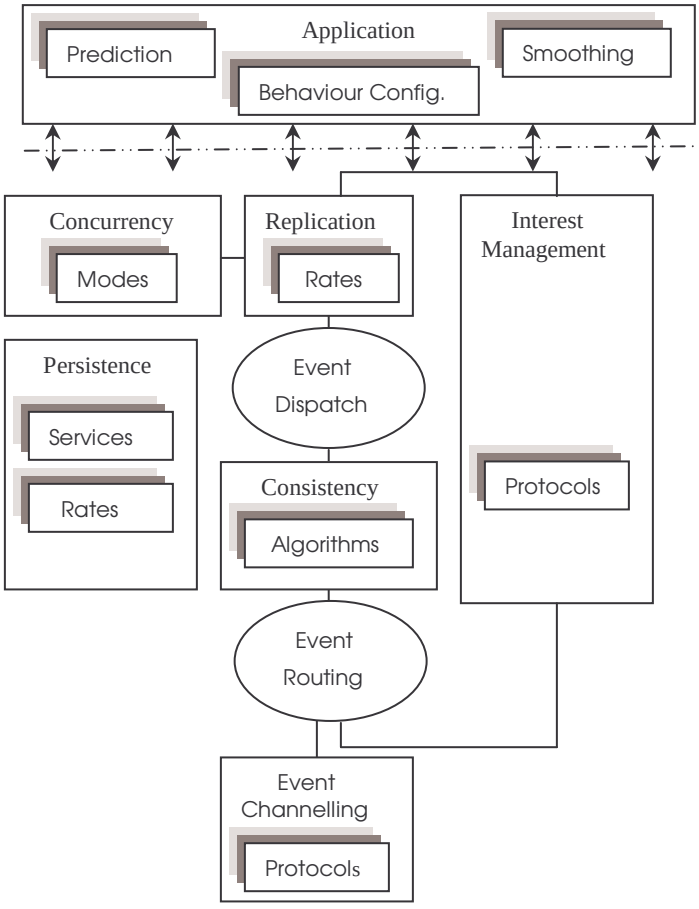

Figure IV Architectural Design

4.1.1 Object and Event Management Bundle

At the Object and Event Management Bundle, five services present run-time pluggable or unpluggable mechanisms as detailed below:

Replication: To provide responsiveness and counter the drawbacks that are commonplace in centralized approaches, a number of networked game applications implement a replicated distribution architecture. OpenPING's Replication service comprises:

Replication Rate with a:

- Standard Rate which replicates data between a node and other peers at an average rate.
- Low Rate which does the replication between a node and its peers at a lower frequency than the standard one.
- High Rate which updates shared data between the master node and replicas at a higher frequency than the average one.

Concurrency: In order to provide acceptable interaction between users that are dispersed over large geographical areas, OpenPING's shared data is replicated amongst participant nodes. Its Concurrency service bundle comprises:

Lock Transfer Mode with a:

- Pessimistic Scheme which blocks all nodes until a lock request is received and is subsequently granted allowing the node to manipulate the object.
- Predictive Scheme in which an object's owning node anticipates requests for ownership based on such properties as orientation, navigation, position and speed.

Persistence: This service is concerned with storage of shared data in stable storage such that it is possible to access the stored data should a need arise.

It comprises:

Service Type with:

- In-memory storage in which shared data persists only for the period that the session is on.
- In-disk storage in which the shared data persists even after a session has been terminated.

Check-point Rates with a:

- Standard rate at which a snap-shot of the shared state of the game session or simulation state set is taken for in-disk or in-memory storage.
- Low Rate which takes a snap-shot of the game session's shared state set at a lower frequency than the standard one above.
- High rate which takes a snap-shot of the game session's shared data set at a higher frequency than the standard one.

Consistency: The fact that consistency of a shared state cannot be guaranteed even when all updates are delivered to all peers implies that there is always the need for consistency algorithms.

It comprises:

Algorithms with:

- Receive-order in which event delivery is in the order that events are received at the consistency service.
- Priority-order algorithm which inserts new events at the correct place in the event queue with reference to event creation time at the application level.
- Total-order which makes use of causal timestamps to ensure a total ordering of events across all simulations.

4.1.2 Application Bundle

Based on an application object behaviour classification, OpenPING's Application service bundle presents instances of application-specific mechanisms. These include the following categories:

Prediction: This involves modeling of deterministic application behaviours at respective nodes in order to compensate for high latency with increased processing by each entity via envisaging the Master object's trajectory in the simulation.

Behaviour Configuration: Implementation of the framework's Application service bundle configures behaviours by dynamically dropping, picking and replacing their attributes depending on external and local load levels or user preferences.

Smoothing: The framework's Application service bundle implements algorithms that are applied to counter jerking visual effects on a moving object's trajectory. This is obviously dependent on the rate at which updates are sent to the node.

4.1.3 Communications Bundle

OpenPING's Communication service bundle comprises three protocols within the Event Channelling service bundle as detailed below:

Reliable Channel: Applications are provided with a simple Application Programming Interface (API) where reliability is a toggle and the network service provides adequate buffers to resend information when reliable messages have not arrived at their destination.

Application Level Framing (ALF) Channel: This type of channel results from a network principle which advocates that the application should help the networking level when transferring data in-between the end points.

Unreliable Channel: OpenPING's unreliable protocol offers a network channel with a rather simplistic transmission of events. In its User Datagram protocol (UDP)-type implementation, the network service provides no buffers at all to retransmit packets that do not arrive at their intended destination.

Behaviours can be broken down into individual constituent parts called Behavioural Attributes (BAs). A Behavioural Attribute is a separable part of the behaviour of an object. Considering motion in a game, *InertiaSlave* (an algorithm that models the deterministic *Inertia* behaviour at the slave simulations) is a BA of the behaviour *Inertia*. It encapsulates a reactive program and can be configured or reconfigured individually using properties/methods/events. A reactive program describes a BA and its associated state.

4.2 Adaptation Management

Adaptation management concerns the observation of objects, decision-making based on observed trends, and the subsequent enactment of decisions through a feedback and control loop [8]. The diagram below presents adaptation management in our architecture.

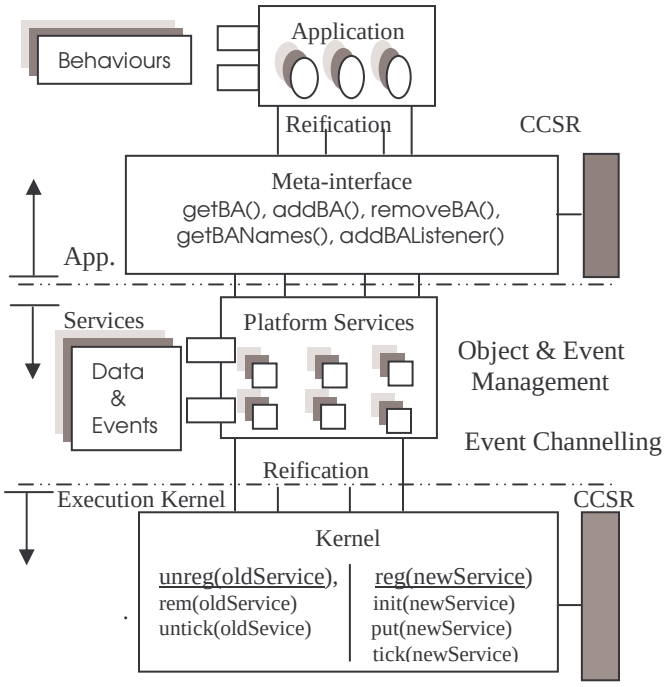

Figure V OpenPING's Adaptation Management

As can be seen from the figure above, the object model (earlier presented in section 3) provides meta-information about itself at two separate meta-meta levels – the first one is at the application bundle which handles shallow behaviours and the second level is at the platform kernel which handles deep and hybrid behaviour.

From an application programmer's viewpoint, the application bundle models both application behaviours alongside a representation of platform behaviours, thus providing a common metaphor for adapting the entire system.

We choose to focus our efforts on Replication, Consistency and Event Channelling services for our experiments since efforts to address scalability, responsiveness and persistence concerns have focused on the Interest Management, Concurrency and Persistence services.

5. EXPERIMENTS AND EVALUATION

From our implementation, we have set up experiments (covering all the behavioural) types that focus on allowing developers to adapt object behaviour at run- time. Our experimental prototype is a simple 'RobotWar' game in which remote users attempt to 'fire' at one another's robots using 'canons'. The primary goals of this experimental work are to evaluate the framework's:

- Use of structural reflection to address *flexibility*, *maintainability* and *extensibility* in networked game applications.
- *Flexibility* in provision of multiple infrastructure mechanisms to address *scalability*, *persistence* and *responsiveness* needs.
- Provision of a reflective model that supports run-time adaptation of application as well as platform behaviours.
- Ease of use and expressiveness with which the game application designer is able to incorporate the platform's mechanisms alongside application-specific behaviour.
- Selective performance metrics and scalability.

Our interpretation models context-specific application (shallow) behaviours alongside standard implementations of platform (deep) and hybrid (shallow/deep) behaviours. Two examples are briefly outlined below.

(Further details can be accessed in the author's PhD thesis currently in preparation).

5.1 Experiment 1 – Hybrid Behaviours

Aim: To enable dynamic causal addition/removal of the ALF Event Channelling protocol.

Implementation: While the system executes, an application switch to *GravityBA* causally activates a switch by the Event Channelling service bundle to *UnreliableEventChannelBA* such that the underlying platform makes up for the additional load at the Application service bundle. Conversely, whenever *GravityBA* is disabled, ALF*EventChannBA* is activated to exploit the information that the application has on the game.

Result: This experiment shows that the middleware adapts to the increase in system load by sacrificing application-semantics' involvement in event delivery. Conversely, it adapts to a decrease in system load by activating reliable event delivery at the platform.

Evaluation: The results of the experiment prove that the framework's reflective model supports run-time adaptation even in instances where behaviours cannot explicitly be referred to as purely platform or application.

5.2 Experiment 2 – Performance Metrics

This experiment evaluates the performance overhead that is directly attributed to the additional code used to realise reflection hence run-time adaptation within the framework. It involves the use of Intel PIII PCs with 128 MB – 256MB memory and 650MHz clock speeds in a 100 Mbps Fast Ethernet Local Area Network (LAN). All the experiments are done on single idle processors and averages (with typical variations measured at ± 2 milliseconds) taken over 100 independent runs.

Aim: To appraise performance metrics and scalability of the OpenPING framework.

Implementation: At start-up, a measure is done on the period of time it takes to load and initialise all services from the platform and start the 'RobotWar' game. Subsequent measurements are made with varying numbers (N) of either Application behaviours or Platform/Hybrid behaviours loaded at the same instant.

Result: It takes an average of 1,735 milliseconds to load the platform and the game at start-up. The total variance between time measurements regarding the configuration or re-configuration of all behaviours during normal operation is 31 milliseconds. Configuring (getting/adding or getting/removing) a single (or two) Application, Platform or Hybrid Behavioural Attribute(s) either at start-up or run-time (during execution) costs 16 ms of execution time while it costs a maximum of 31 ms of execution time to load as many as 10 behaviours at the same instant.

The contribution this makes towards attainment of the recommended threshold for effective end- to-end lag in propagation of multimedia data (100 – 300 ms) [5] is not significant.

Below is a graphical representation of loading time (ms) against Behaviours (N) at start-up.

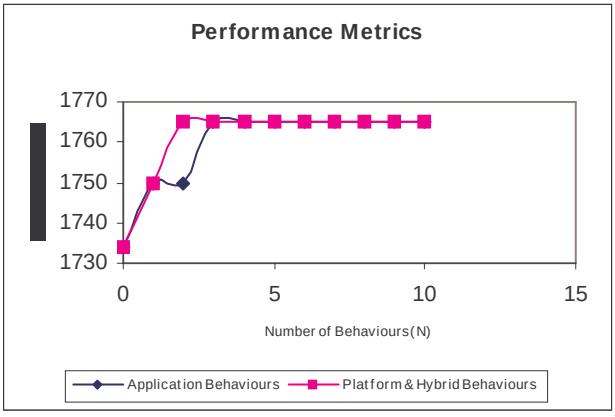

Figure VI Execution time for configuration of Platform Services and application-specific BAs at start-up.

Evaluation: The figures above give credence to the fact that at just about 1% (of the total execution time) as an overhead incurred by the framework, incorporation of run-time adaptation through structural reflection offers tangible benefits.

Since as many as 10 Behavioural Attributes (BAs) are configured at the same instant without an exponential increase in execution time, the approach taken fully meets scalability demands in next generation networked game service platforms.

6. CONCLUSION

This paper has outlined the need for dynamic adaptation as a means to achieve better flexibility, maintainability and extensibility and also offer support in a flexible way for the run-time incorporation of scalability, persistence and responsiveness techniques. Incorporating dynamically evolving application-specific wishes by making modifications (on the middleware or application) at compile-time is not ideal especially if the application involves real-time interaction and requires round-the-clock availability. To support dynamic adaptation, this paper has detailed how our framework facilitates not just the co-existence of multiple alternative infrastructure mechanisms but additionally, rather than applying a single mechanism to all environmental scenarios, mechanisms can be tested, replaced, reconfigured or dropped at the application level in the same manner that behaviours in the application are.

Hence we argue that in distributed virtual reality, the use of reflection at the application level to design a meta-interface through which internal managers monitor and adapt platform and application behaviour dynamically is the way forward in the design of next generation networked games platforms.

REFERENCES

[1] Birman K., R. Cooper, T. Joseph, K. Marzullo, M. Makpangou, K. Kane, F. Schmuck and M. Wood, *"The ISIS System Manual, Version 2.1"*, Cornell University, Sep. 1990.

[2] Frecon Emmanuel, Marten Stenius, *"DIVE: A Scalable Network Architecture for Distributed Virtual Environments"*, Distributed Systems Engineering, 5(3), pp. 91-100, 1998.

[3] Frederic Dang Tran, B. Dumant, F. Horn, J.B. Stefani, *"Jonathan: an open distributed processing environment in java"*, Middleware '98, IFIP International Conference on distributed Systems Platforms and Opens Distributed Processing, Lake District, UK, Sep. '98.

[4] Frederic Dang Tran, Anne Gerodolle, *"An Object-oriented Framework for Large-scale Networked Virtual Environments"*, Springer-Verlag, In Proceedings of the 6th International Euro-Par Conference, Munich, Germany, Sep. 2000.

[5] ITU90, *"Effect of propagation delays on communication quality"*, International Telecommunications Union (ITU) SG12, Feb. 1990.

[6] Lea Rodger, Yasuaki Honda, Kouchi Matsuda, *"Virtual Society: Collaboration in 3rd spaces on the Internet"*, The Journal of Collaborative Computing, 1997.

[7] Okanda, P., Blair, G., *"Analysis of Techniques used in Distributed Virtual Environments"*, Internal Report Nº. MPG-02-01, Computing Department, Lancaster University, Nov. 2002.

[8] Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Qulici, A., David S. Rosenblum, D.S., Wolf, A.L., *"An Architecture-Based Approach to Self-Adaptive Software"*, IEEE Intelligent Systems, Vol. 14, No. 3, pp. 54-62, May/June 1999.

[9] Purbick James, *"Continuously Available Virtual Environments"*, PhD. Thesis submission, Nottingham University, UK '01.

[10] Rao, R., *"Implementational Reflection in Silica"*, Proceedings of ECOOP '91, Lecture Notes in Computer Science, P. America (Ed), pp. 251-267, Springer-Verlag, 1991.

[11] Smith, B.C., *"Procedural Reflection in Programming Languages"*, PhD. Thesis, MIT, Available as MIT Laboratory of Computer Science Technical Report 272, Cambridge, Mass., 1982.