OpenPING: A Reflective Middleware Platform for construction of Adaptive Virtual Reality Applications

Paul Okanda, Gordon Blair, Nikos Parlavantzas
Distributed Multimedia Research Group,
Computing Department,
Lancaster University, Bailrigg, Lancaster LA1 4YR, UK.

Tel: +44 1524 65201, ext: 93315
Fax: +44 1524 593608
Email: {okanda, gordon}@comp.lancs.ac.uk, n.parlavantzas@lancaster.ac.uk

Abstract

The emergence of collaborative virtual world applications that run over the Internet has presented Virtual Reality (VR) 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 virtual world applications be aware of and adapt to frequent variations in their context of execution. In this paper, we argue that the use of structural reflection offers great potential for the design of flexible real-time interactive Distributed Virtual Environments (DVEs).

Keywords

Distributed Virtual Environment (DVE), Virtual Reality (VR), Reflection, Object Behaviour, Adaptation.

1. Introduction

Recent research has been aimed at developing distributed platforms that can support real-time applications running on the public internet. This has proved extremely challenging, particularly in massively multi-participant applications where thousands of users potentially interact in real-time with each other and with thousands of autonomous entities using uncontrolled network and local (processor, memory) resources. Requirements for such systems include *scalability*, *persistence*, *responsiveness*, *flexibility*, *maintainability*, and *extensibility*.

The main focus of research on platforms that support real-time interaction has been on the first three capabilities and, as a result, a number of techniques both at the platform and application level have emerged.

To improve scalability, existing published works propose a wide range Virtual World partitioning approaches from static coarse-grained partitions [Frecon98] to interest management (perception-based) approaches (VS) [Lea97].

To address persistence requirements, some DVE platforms such as Continuum [Frederic00] implement mastership transfer within peer-to-peer architectures.

To provide support for responsiveness, researchers in real-time interactive VR systems have attempted to implement fully distributed architectures together with multicast grouping of clients, e.g. DIVE [Frecon98]. (A detailed analysis of techniques used in contemporary DVEs can be accessed in [Okanda02a]).

In contrast, there has been much less effort on addressing the *flexibility*, *maintainability* and *extensibility* requirements of contemporary DVEs. 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*.

This paper is structured as follows:

Section 2 presents a definition of reflection. Section 3 then provides an insight into our system design while implementation details and an overall architecture are covered in section 4. Section 5 briefly mentions some experiments done and their evaluation. Section 6 then presents related work and finally, section 7 concludes the paper.

2. Background on Reflection

The conceptual origin of reflection could be traced to Smith [Smith82] who introduced it thus:

'In as much as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of that world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representations of its own operations and structures'.

This insight triggered an initially limited body of research in programming languages such as Lisp and a few others in the object-oriented community. Subsequently, this research diversified into areas such as distributed systems [McAffer96] in which contemporary research emphasis has been on architecting middleware platforms that are geared towards accommodating a wide variety of requirements imposed by both applications, mobility and underlying environments.

For the purposes of this paper, we provide a simple context-specific definition of structural reflection in DVEs as;

'a design principle that allows a Virtual World to have a representation of itself in a manner that makes its adaptation to a changing environment possible'.

3. System Design

Essentially, the motivation for this work is to incorporate flexibility, maintainability and extensibility into DVEs that support real-time interaction. The next section provides details of our design.

3.1 The Object Model

Reflection per se does not support flexibility, incrementality or ease of use as this only comes about through the additional application of object-orientation.
This provides the drive for our use of an object-oriented approach in our design.
In our object model, 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 the DVE terrain) contain all elements except the set of behaviours. This object model is then implemented with an appropriate Meta Object Protocol (MOP) as defined in section 4.3 below. Firstly, however, we consider the role of behaviour in DVEs.

3.2 The Role of Behaviour

The design of DVEs seeks to model VR applications around various interpretations of reality. Real life artefacts exercise their behaviour to perpetuate their significant subsistence. For example, human beings exercise their 'eating' behaviour without which they would not meaningfully exist.
Behaviour is also used to describe artefacts in real life. For instance, mammals nourish their young with milk secreted by mammary glands. The phrase 'nourish their young with milk secreted by mammary glands' is an observable behaviour that defines the existence of human beings as mammals.
The fact that behaviour forms an integral part of the existence of real life artefacts gives it a crucial role in our attempts to model them. In VR, behaviour provides a handle in the capture (simulation) of the real world phenomena and their run-time adaptation policies/mechanisms.
We therefore define behaviour as the way in which the state of an object's attributes changes over time. For instance, an object may have an attribute called 'location'; as it moves around, its location changes. The way in which its location changes over time is its behaviour.
We look into an object model that has three categories of associated behaviours:

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

3.3 The Meta-model

We adopt the object model described in sub-section 3.1 above and use techniques that allow behaviours to be encoded and subsequently be evolved and adapted at run-time.
In particular, we define a meta-interface (Meta Object Protocol) which essentially offers structural reflective capabilities on application objects with operations that:

- *discover* the internal details of an object in terms of attributes, behaviours etc,
- *insert* a new attribute, behaviour or rendering,
- *delete* an existing attribute, behaviour or rendering or
- *replace* an existing attribute, behaviour or rendering.

This MOP can then be used for adaptation over the object model described earlier.
Adaptation is essentially the alteration of the underlying implementation of a system in order to suit the needs of its fluctuating execution environment. These fluctuations range from user subjectivity to the system's infrastructural setting. We use the system design shown above to achieve adaptation as described in the next section.

4. Implementation

4.1 Overall Architecture

A platform has been implemented based on the above design. The platform's architecture offers dynamism in DVEs through exploitation of application-specific semantics and run-time execution environment awareness. The architecture provides the application designer with access to application objects as well as mechanisms encapsulated in six service bundles within a middleware platform called OpenPING[1]. The service bundles, each with its own set of pluggable mechanisms are: *Concurrency, Replication, Interest Management, Persistence, Consistency* and *Event Channelling*.
The diagram below illustrates the overall architecture of the platform.

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

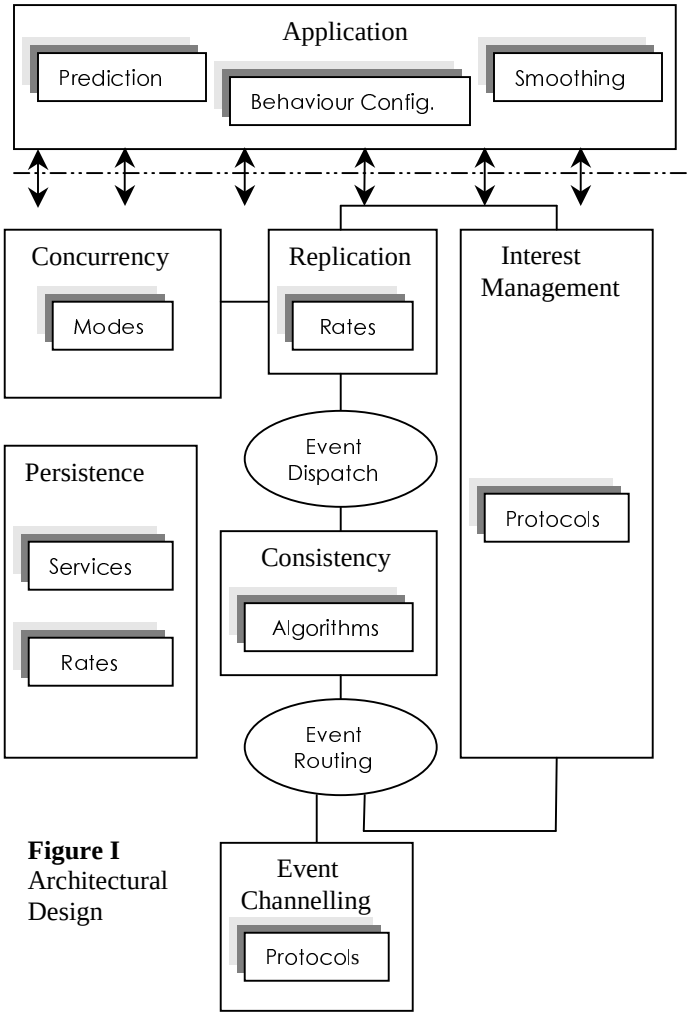

Figure I Architectural Design

The rationale for the architecture above has a basis in the earlier identified need for incorporation of flexibility and run-time adaptation in contemporary DVEs. This must be considered over a set of services and mechanisms with policies defined to manage their dynamic configuration over an execution kernel.
Table I below shows the *Object and Event management Layer* in which five service bundles present run-time pluggable mechanisms.

Service	Mechanism & Details
Concurrency	Lock Transfer Mode [standard or predictive] – normal change of mastership & subsequent lock transfer of locks or predictive anticipation of mastership.
Replication	Rate [standard, high or low] – variable provision of multiple instances of the same entity at different nodes via periodical updates.
Persistence	Service-type [in-memory or in-disk] – local (processor and memory) resources determine when to switch. Check-point Rates [low, high or standard] – variable snap-shot taking of the world.
Consistency	Algorithm [receive-order, priority-order or total-order] – receive order using First-In-First-Out (FIFO) in satisfactory network conditions, priority-order with reference to event creation time and total-order when strong consistency is a major concern.
Interest Management	Protocol [spatial or publish-subscribe] – spatial based protocols when network conditions are perfect and publish-subscribe protocols when there's need to filter event transmission.

Table I Mechanisms at the Object & Event Management Layer

The *Application Layer* presents instances of application-specific mechanisms. Examples that apply in our experimental context are categorised as presented in **Table II**.

Mechanism	Details
Prediction	[on or off] – modelling deterministic behaviours at nodes to compensate for high latency with increased processing by each entity through envisaging the Master avatar's trajectory.
Behaviour Configuration	[drop, pick or replace] – dynamic dropping, picking or replacement of behaviours depending on Local/External load levels or User preferences e.g. replacement of rich text with plain.
Smoothing	[simple, standard or complex] – algorithms applied to counter jerking visual effects on the avatar's trajectory depending on the rate at which updates are sent to the node.

Table II Mechanisms at the Application Layer

Finally, the *Communication Layer* comprises the Event Channelling service bundle with one mechanism as shown in **Table III**.

Mechanism	Details
Protocol	[reliable, unreliable or Application Level Framing] – reliable channelling used to relay events that require high levels of reliability, unreliable channelling used when high system load levels presents a bigger problem than reliability and Application Level Framing (ALF)[2] when local resources are available and some form of application control over packet loss detection and recovery is important.

Table III Mechanisms at the Communication Layer

We choose to focus our efforts on Replication, Consistency and Event Channelling service bundles for our experiments since efforts to address scalability, responsiveness and persistence concerns have concentrated on the Interest Management, Concurrency and Persistence service bundles.
Each mechanism is represented as a pluggable behaviour at the application level. Behaviours can be broken down into individual

[2] A networking service protocol model that explicitly includes an application's semantics in the design of that application's protocol [Floyd 90].

constituent parts called Behavioural Attributes (BAs). We define a Behavioural Attribute (BA) as a separable part of the behaviour of an object. It encapsulates a *reactive program* and can be configured individually using properties, methods or events. We use a reactive programming approach to avail a flexible paradigm for encoding reactive systems, especially those which are dynamic since it provides application programmers with a fine control over concurrency, event broadcast and several primitives for gaining fine control over program execution.
More specifically, we use a tool called Junior (Jr).
In the next sub-section, we define the reactive programming paradigm and Junior (Jr).

4.2 Reactive Programming

Reactive programming is a process which involves the encoding of reactive instructions. Since active objects have their own specific behaviour and react continuously to events occurring in their environment (interactions with other objects or time progression), programming active objects (e.g. avatars) in a shared virtual world is essentially a form of reactive programming.
Software systems that are used in reactive programming have some inherent characteristics [Hazard99]. They (software systems) are event-based, parallel but thread-less and reactive.
Junior (Jr) is a Java-based kernel model for reactive programming which defines concurrent reactive instructions communicating using broadcast events [Hazard99].
Programming in Jr is essentially a four-stage process that involves:

1. declaring a reactive machine
2. writing a reactive instruction
3. dropping the program into the reactive machine
4. running the reactive machine.

Below is an example to illustrate the above process. It runs *Receive-OrderBA*, a platform (deep) behaviour from the Consistency service bundle whose prime purpose is to order events FIFO from the Object and Event Management Layer to the Application Layer.

```
import junior.*;

public class Behaviour
{
  public static void main(String[] args){
      Machine machine[1] =
  Jr.Machine[3](Jr.Loop(Jr.Seq(Jr.Atom(new
      ReceiveOrderBA()),Jr.Stop()[2])));
      machine.react()[4];
  }
}
```

4.3 Adaptation Management

Adaptation management concerns the monitoring of objects, the decision making based on observed trends, and the subsequent enactment of the decisions through a feedback and control loop.
Our meta-interface drives such behavioural changes as addition/removal at run-time of pluggable/unpluggable purely application behaviours, purely platform behaviours and hybrid behaviours.
It can be used by external entities with respect to the object as external adaptation or by a Behavioural Attribute (BA) of the same object as an instance of self-adaptation.

We perform various instances of both:

- coarse-grained adaptation at run-time for instance in addition/removal or replacement of algorithms earlier mentioned in the Consistency service bundle or protocols in the Event Channelling service bundle.
- fine-grained adaptation for instance in configuration of rates used within the Replication service.

The diagram below gives a visualisation of adaptation management in our architecture.

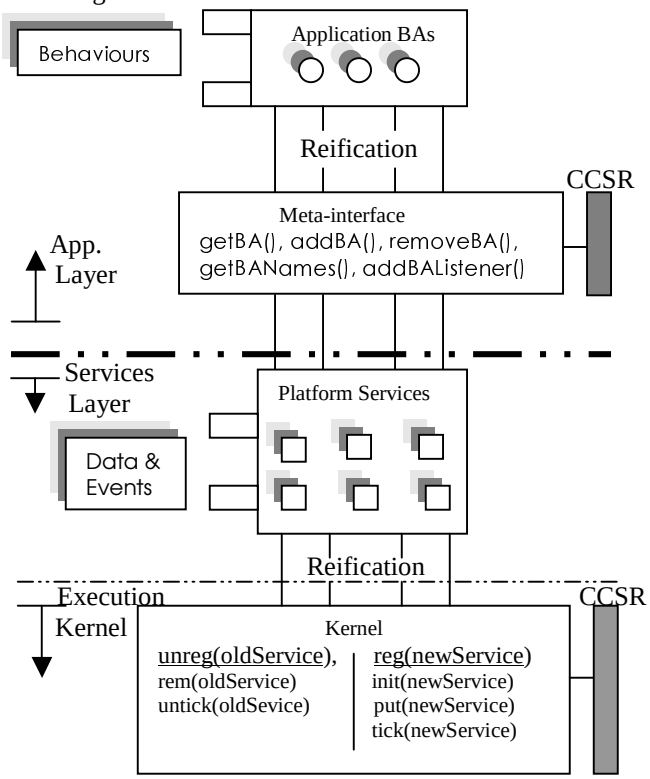

Figure II Adaptation Management

The *Application Layer* models both application behaviours and also a representation of system behaviours, thus providing a common metaphor for adapting the system. Run-time adaptation of the application-specific behaviours occurs within this layer while the more generic system behaviours adapt via configuration and reconfiguration of platform services. In both cases though, this is modelled as changes in behavioural attributes. To support this, the meta-interface offers operations to discover, insert, delete and replace both application and system behaviours via such constructs as `addBA(), getBA()` etc.
The *Services Layer* comprises the entire Platform's service bundles mentioned earlier in **Figure I.** These are handled in form of data structures and events. Its Execution Kernel offers a Causally Connected Self Representation (CCSR) of the Platform services and a reification that enables transparent (from the application programmer's viewpoint) unregistration (of an old service) and registration (of a new service). Just like application-specific behaviours, the run-time configuration of platform services is done in the form of operations that the *Application Layer's* meta-interface offers. Hence invoking these operations at the *Application Layer* triggers corresponding actions within OpenPING's execution kernel to unregister 'old' services and register 'new' ones dynamically.

5. Experiments and Evaluation

We have set up a series of experiments that focus on allowing developers to dynamically adapt object behaviour. The experiments cover a range of application behaviours, platform behaviours, hybrid behaviours and performance metrics. We present two of them (on platform behaviours and performance metrics) in sub-sections 5.1 and 5.2 below.
(Full details of the experiments, their results and evaluation can be accessed at [Okanda02b]).

Our experimental prototype is a simple 'RobotWar' game in which remote users attempt to 'fire' at one another's robots using 'canons'. In the game, each user has ownership of a single robot (replicated at all remote sites) which can move around and 'holds' a 'canon' that 'fires' at the rest of the users' robots at a key-press. The challenge is to evade all the opponents' 'missiles' and at the same time 'shoot down' their robots. In the 'RobotWar' game, our interpretation models context-specific application (shallow) behaviours alongside standard or generic implementations of platform (deep) and hybrid (shallow/deep) behaviours.

5.1 Experiment 1 – Platform Behaviours

Aim: To drive run-time causal addition/removal of the Consistency service algorithms: Receive-order, Priority-order and Total-order.
Implementation: *Receive-orderBA* uses simple FIFO event ordering and as such is good enough in satisfactory network conditions. *Priority-orderBA* is used whenever network conditions (monitored via disparities in Master and Slave object positioning) are unsatisfactory.
The system adjusts to the increase in system load by sacrificing strict event ordering (that is activated by *Priority-orderBA*). Conversely, the system fine-tunes itself to a decrease in system load by activating strict event ordering at the platform.
Total-OrderBA's use is not illustrated in this experiment but it is worth noting that it's implementation suits simulations in which very strict consistency is of paramount importance.
Result: The framework's execution is such that *Priority-orderBA*'s addition is causally triggered at the instant the application-level behaviour *InertiaSlaveSimpleBA* is added and the behaviour *Receive-orderBA* causally activated whenever *InertiaSlaveComplexBA* is executed. This evidence reveals how much like application behaviours, platform behaviours can flexibly be configured run-time to conform to fluctuating network and system resource availability.
Evaluation: This experiment illustrates how OpenPING's flexibility facilitates adaptation to fluctuations in load levels and network conditions within the system's execution environment. It shows how the framework's provision of a MOP avails a set of meta behaviours (accessible at the application level) that support the designer in his/her choice of implementation from a variety of mechanisms to suit different execution conditions.

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 involved 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 were 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. To quantify the impact Behavioural Attribute (BA) configuration has on OpenPING's performance, subsequent measurements are made with varying numbers (N) of application behaviours or platform/hybrid behaviours loaded at the same instant. Measurements are also made to quantify the period of time it takes to configure/reconfigure BAs during normal operation (i.e. after start-up).
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 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 application behaviours at the same instant.
Loading a single platform or hybrid Behavioural Attribute (BA) at start-up or run-time costs 16 ms while it costs a maximum of 31 ms of execution time to load 10 of them 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) [ITU90] is not significant.
Below is a graphical representation of loading time (ms) against behaviours (N) at start-up.

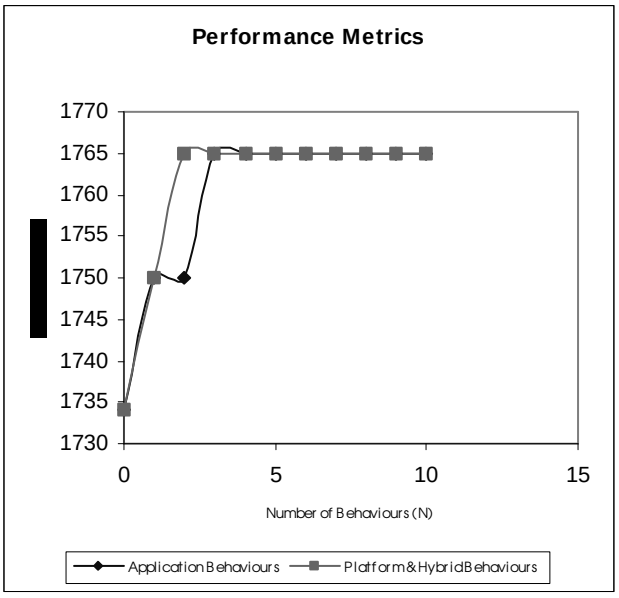

Figure III Execution time for configuration of Platform Services and application-specific BAs at Start-up

Evaluation: The figures above lend 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.

The fact that as many as 10 Behavioural Attributes (BAs) are configured at the same instant (at start-up or during execution) without an exponential increase in execution time proves that the approach taken fully meets scalability demands in next generation DVEs.

Overall, their (experiments') results:

- demonstrate how OpenPING's Meta interface offers support to the designer in his/her choice from a variety of mechanisms in a flexible way.
- show how OpenPING's multiple infrastructure mechanisms co-exist to enable run-time configuration via policies that the DVE designer defines at the Application Layer.
- prove OpenPING's provision of support for dynamic (as opposed to compile-time) adaptation of application as well as platform behaviours either at start-up time or during execution.
- epitomize the simplicity, ease and expressiveness with which the DVE designer incorporates OpenPING's mechanisms alongside application-specific behaviours.
- prove that the overhead incurred in execution time is not too big a price to pay in order to avail the full benefits of flexibility and scalability.

6. Related Work

MASSIVE [Greenhalgh95] is an experimental prototype whose need for incorporation of run-time adaptation to suit real-time interaction is clearly evident.
DIVE [Frecon98] has weaknesses resulting from its assumption that networks have low-loss and reasonably high band-width hence low latency for collaborative manipulation. In early versions of DIVE, the ISIS toolkit [Birman90] uses a multicast protocol to distribute changes and set locks. All nodes are guaranteed to have seen the same sequence of events, which while good for system integrity, limits scalability. On the other hand, in the absence of the ISIS toolkit, consistency guarantees which inevitably improve interactive manipulation especially in environments with high network latencies are non-existent.
Continuum [Frederic00] offers an array of service options but these are essentially compile-time and do not come with an interface or execution kernel that supports run-time adaptation of mechanisms.
CAVE [Purbrick01] investigates persistence in DVEs by associating behaviour with platform services in much the same way the application level provides a handle on objects. It however has a limited scope as it only tackles the issue of persistence in continuously available large-scale virtual environments.

7. Conclusion

This paper has outlined the need for dynamic adaptation as a means that achieves better flexibility, maintainability and extensibility and also supports in a flexible way, the run-time incorporation of scalability, persistence and responsiveness techniques.
Application-specific wishes concerning the configuration of platform services inevitably evolve dynamically. Incorporating these 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, configured or dropped at the application level in the same manner that behaviours in the application are. Hence, we argue that use of reflection is the way forward in the design of next generation DVEs.

References

[Birman90] 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, September 1990.

[Frecon98] Frecon E., Marten Stenius, *"DIVE: A Scalable Network Architecture for Distributed Virtual Environments"*, Distributed Systems Engineering, 5(3), pp 91-100, 1998.

[Frederic98] 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, September '98.

[Frederic00] 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, September 2000.

[Greenhalgh95] Greenhalgh C., Benford S., *"MASSIVE: A Distributed Virtual Reality System incorporating Spatial Trading"*, 15th IEEE International Conference on Distributed Computing Systems (ICDCS '95), 1995.

[Hazard99] Hazard Laurent, Jean-Ferdy Susini, Frederic Boussinot, *"The Junior Reactive Kernel"*, Research Report, CNET/INRIA RR-3732, July 1999.

[IEEE Computer Society] *"IEEE Standard for Distributed Interactive Simulation-Application Protocols"* (IEEE 1278.1-1995), IEEE Computer Society, 1995.

[Kiczales91] Kiczales, G., J. des Rivieres, D.G. Bobrow, *"The Art of the Metaobject Protocol"*, MIT Press, 1991.

[McAffer96] McAffer, J., *"Meta-Level Architecture Support for Distributed Objects"*, In Proceedings of Reflection 96, G. Kiczales (Ed), pp 39-62, San Francisco.

[Okanda02a] Okanda, P., Blair, G., *"Analysis of Techniques used in Distributed Virtual Environments"*, Internal Report N°. MPG-02-01, Computing Department, Lancaster University, November 2002.

[Okanda02b] Okanda, P., Blair, G., *"Experiments and Evaluation of the OpenPING Framework"*, Internal Report N°. MPG-02-02, Computing Department, Lancaster University, November 2002.

[Purbrick01] Purbrick James, *"Continuously Available Virtual Environments"*, PhD. Thesis submission, Nottingham University, UK, October '01.

[Smith82] Smith, B.C., *"Procedural Reflection in Programming Languages"*, PhD Thesis, MIT, Available as MIT Laboratory of Computer Science Technical Report 272, Cambridge, Mass., 1982.