Abstract - In the digital world new and emerging devices are continually appearing on the market. MP3 players, mobile phones, and notebooks, including game consoles are common place with many enjoying networking capabilities. This has provided a platform on which new and innovative usage scenarios can be realised. For example, content can be easily created on a mobile phone and watched on a television or incorporated into the games we play. In principle networks and the devices they consume now make it easier to move content between network services and devices. In this paper we build on this idea where content is not something that resides in a location in which it was created or placed but something that is virtually connected to owners’ content. This allows content to be metaphorically dragged between content services and devices through highly interconnected networks depending on what device we use or which virtual environment we inhabit. This has already been seen in part in gaming and virtual world developments, where pre-existing content can be modified or user-generated content is used to form part of game and virtual world environments. The principle flaw being that any effort and content remains within the environments used to create them. The challenge is to harness this effort where content can be created independently of environments, thus freeing users to choose where content should reside. In this paper we propose one possible solution that uses semantic annotations to describe and integrate content visualisations and behaviours into heterogeneous environments. We demonstrate our findings using a working prototype.

Keywords: Networked Virtual Environment, Content Sharing, Rules Engine, Dynamic Scripting.

I. INTRODUCTION

Today nearly every device has the ability to connect to the internet. Even game consoles are networked devices designed to link users to huge communities within different virtual worlds. Furthermore, online virtual environments such as Second Life have undoubtedly pre-empted widespread adoption of socially-inspired gaming. This is seen as an Internet phenomenon that may form the bases for the next generation Internet, which will support many 3D functions. Given this phenomenon, users now consider virtual friends and their avatars just as important as those in the real world. Coupled with the ability to more easily socialise in virtual environments we also see the emergence of organic commerce where users readily share virtual objects for use in those environments or products that manifest themselves in the real-world as physical products.

This only paints part of the picture where less conventional devices are now also beginning to form part of the Internet and are being consumed within virtual environments. For example, mobile devices such as mobile phones are now enjoying networking capabilities to seamlessly form part of larger networks. This opens up new and exciting opportunities where the benefits of interoperability between networked devices and virtual worlds are obvious; for example, devices can be controlled remotely because of improved interaction and; secondly, both real-world devices and virtual worlds can freely share functionality and content between the two. This has helped blur the gap between the real and the virtual. Interpretations can be made about content shared, which may include conventional multimedia as well other digital content such as guns, and life for game play. This allows a more intimate link between heterogeneous games where such interpretations form the basis for visual renditions as well as the ad hoc generation of behaviours those objects being shared support. Whilst understanding the effects a car crash may have in a driving game, such interpretations allow the car to inhabit a world in which the concept of a car is not necessarily understood, but where behaviour mappings allows a comparison to be made between the effects of a crash and for example, that of being shot at. This not only makes games more flexible, but it also provides a basis for more interesting virtual environments not yet seen.

This is a challenge where sharing content and devices and their usage scenarios in the real world does not entirely map onto sharing content and devices and their usage scenarios in the virtual world. We have a level of intelligence that makes...
this process easy, which is not shared with computers. This is a problem well documented within the Artificial Intelligence domain, however, aspects from Artificial Intelligence can be utilised. We borrow from many of the successes seen in the Semantic Web, more specifically the use of semantic descriptions to describe content visualisations and behaviours (the interaction between objects across different virtual environments). Our novel contribution resides in the fact that content is not something that resides in a location in which it was created or placed but something that is invisibly connected to content owners. Our framework allows content to be metaphorically dragged between content services and devices through highly interconnected networks dependent on what device we use or which environment we inhabit. Therefore, we propose a workable system prototype in which semantically described content visualisations and behaviours can be used within interconnected worlds that comprise devices and virtual environments.

The remainder of this paper is structured as follows. In Section II we introduce the background and related work. In Section III we provide an overview of our proposed framework before presenting implementation in IV and our conclusions in Section V.

II. BACKGROUND AND RELATED WORK

Massively multiplayer online games attract huge numbers of players and this is set to increase. Utilising Internet communications, games have transcended large geographically dispersed continents, countries, and cities, and embraced many different cultures. Coupled with unconventional gaming devices, such as mobile phones, mass convergence between real and virtual world activities and different social activities both in our real lives and in social networks has resulted in games that adopt multi-dimensional functions [1]. This has changed how users view and play games. For example, many games such as *Planetside* [2], *The Sims Online* [3] and *EVE Online* [4], are dependent on network communications. None more so than the game *World of Warcraft*, which became the fastest selling PC game in North America in 2004-2005 and in 2006 was reported to have 6 million subscribers worldwide [5]. This is undoubtedly a phenomenon that has the capacity to create an overlay community comprising users globally in ways never achieved before and a new world, where people consciously exist, has manifested itself.

From these developments we can see that multiplayer gaming clearly provides significant benefits, be they commercial, technical or social, over single-player games through the use of networking. Nonetheless, its widely used client-server architecture enforces a number of limitations. Most notably, game play and enhancements must be carefully controlled through centralised gaming servers. This has resulted in expensive investment, bottlenecks, central points of failure, and the inability to appropriately react to real-time changes in large virtual worlds. Gamers are tied to games through proprietary software and hardware installations. User interactions do not affect strategic developments and games do not support self-management capabilities to extend functionality beyond those they have been pre-programmed with.

This recent development has lead to shifts within the gaming industry, where increasing access to game engines, software development kits and level editors have allowed games to be changed more easily. This phenomenon – known as modding – marginally alleviates some of the limitations discussed above [6-8]. Although modding provides a means of adapting and evolving games, it is restricted to more technically savvy users, such as software developers, rather than people who simply just play games. Furthermore, mods are tied to specific games. For example, a mod developed for the *unreal engine* will be incompatible with the *quake engine*.

Some researchers suggest that distributed technologies in conjunction with middleware may relieve many of these difficulties, however it is generally accepted that more research is required to establish a suitable architecture [9].

Whilst modding has been discussed from a game coding perspective, mods may also exist as part of or within the game itself. Communities such as *Second Life* [10] are heavily reliant on users shaping the virtual environment, extending the concept of multi-user dungeon (MUDs) into realistically rendered virtual worlds [11]. Graphical objects of any description can be developed and added to the virtual world, which can then be shared or sold between avatars’ within that world. Modifications to the environment (e.g. land) can be made and buildings can be constructed. This differs somewhat from conventional modding in that all modifications take place within the virtual world. However, there is no mechanism to allow the objects created in *Second Life* to be shared and distributed amongst other online games and a better approach could be used to expose these modifications so that they can be utilised universally.

Addressing this requirement needs alternative platforms where the visualisation and operational capabilities of objects can be manipulated externally to the applications in which they are shared and used. New and novel platforms will require intelligent modules capable of understanding and making interpretations about visualisations and the behaviours objects have. Merabti et al. [12] already provide mechanisms to plug services into the network that can act as helper functions, which we exploit in our work. For example object visualisations serialised in the COLLADA format located in a source environment, such as *Second Life*, can be converted into an OBJ format and passed to a target environment that is capable of using this format. Behaviours are slightly different in that it is the target environment and its specifics that are best suited to dynamically creating such behaviours. Initially objects being projected into target environments (objects transmitted over the network and loaded into another environment), would describe their functions semantically. One possible way of doing this could be to apply advances made within the Semantic Web [13] and the tools they use such as OWL-S [14]. These semantic descriptions could be
used as input into services equipped with rule engine capabilities. When semantically matched rules fire [12] they would generate dynamic scripted behaviours, serialised using script languages such as Ruby, Python, Lua or JavaScript, and would be projected alongside the visualisation data into the destination environment.

There are numerous reasons for incorporating scripting languages [15] into games and virtual environments. The productivity benefits, the separation of data and logic, the scalability of the architecture, and the ease with which it can be achieved means that scripting languages should definitely become part of games programmer’s toolkit. Scripting is best utilised for the control of data and behaviour, and as seen by the numerous commercial titles that have chosen to script, this technique can have many non-technical benefits over hard-coding, such as the flourishing modding communities surrounding games of varying genres.

There are many options for incorporating scripting into games and networked virtual environment applications, by utilising an existing solution, or developing a custom bespoke tool. The bespoke tools can be very powerful, and provide the developer with the ultimate control over their solution, but also involve an extremely large amount of development to create them from scratch, putting this method in conflict with the productivity gains made from utilising scripting. When adopting a generic scripting language approach, all four of the languages mentioned provide an adequate range of language features, high level bindings to other languages, built in support functionality, and usage portability. The choice will be application dependent e.g. using Lua if speed and memory footprint are crucial, Python if existing functionality must be utilised, or Ruby for maintaining an Object-Oriented approach.

In the remainder of this paper we build on these advances and propose a framework capable of dynamically interpreting and creating scripted behaviours for object capabilities, which can be migrated, along with object visualisation data, across networked heterogeneous environments.

III. FRAMEWORK OVERVIEW

Existing work on game object/character behaviour modelling can be generally classified into two approaches; microscopic and macroscopic. Most computational models for object/character modelling and simulation adopt the microscopic approach were each individual agent is equipped with a set of decision rules to determine what to do in the next time step.

In the proposed system architecture a two-level model has been adopted. The lower level is used to model individual behaviours, and the top level is used to represent object/character visualisation and interaction. This two-level architecture is a natural reflection of the interaction amongst objects, and between an object and a device in real-life situations. An interaction can emerge amongst individuals and might take into account environmental factors.

Individuals involved in this emerging process may change their behaviours after an interaction is formed. When an object/character joins the new environment, the behaviour of the individual in the new environment will be determined by both the environment model and the object/character behaviour model.

A. Semantic Matching and Generation of Behaviours

Behaviour ontologies allow objects to be semantically annotated, which can be embedded within object advertisements. The same ontologies can also be used to describe object requests. We currently use three main behaviour ontologies; the Behaviour Ontology; Behaviour Profile; and the Behaviour Process Model [48]. Behaviour ontologies allow objects to be described at an abstract level in terms of Inputs, Outputs, Preconditions, and Effects (IOPEs). The IOPEs form explicit relationships between the different ontologies, which are in turn mapped into signatures (method names, parameters and return data including type information). In this way the Behaviour ontologies and the behaviour interface provide a mechanism to link semantic descriptions to possible implementation solutions to describe how behaviours are generated. This process helps independent behaviours offered by objects to be dynamically created or discovered, and executed with little or no human intervention.

Using these principles, functions (the operational capabilities objects support, for example, a weapon in game can provide functions to reduce the life of objects, which can be either used locally to create high-level scripts (object behaviours dynamically generated) or discovered and used via the network. Looking at high-level functionality, Figure 1 shows how a weapon can be created in an environment (an object behaviour that does not exist but rather emerges through the simulation of another object behaviour) were assumptions can be made between being hit by a vehicle and being shot by a weapon.

Using the concept of IOPEs, object behaviours can be dynamically generated by matching similarities between the object request and the behaviour ontologies. Behaviour ontologies, in conjunction with domain ontologies used by the target environment the object is being projected into, are matched through a one to one mapping or semantically if vocabularies are syntactically different but semantically equivalent.
Looking at Figure 1, the user submits an object request to the target environment (step 1). Upon receiving the request the behaviour matcher in the target environment begins by iterating through the Behaviour Profiles for each behaviour it provides (step 2). Using the domain ontology, the IOPEs in the Behaviour Request are matched with IOPEs in the Behaviour Profile. If exact matches are found then the process simply moves onto the next IOPE. However, if there are syntactic differences, the two terms are passed to the domain ontology [12] (step 3 and 4). If a relationship exists between the two terms a match has been found that semantically links the two terms together.

For example, if we were trying to project a weapon, such as a gun, into an environment typically designed to have lots of cars then we would have to make an interpretation about what fire means and the effect it has on objects in that environment. For example, looking at Figure 1, we could use the domain ontology to work out this effect through the semantic links, i.e. firing something causes damage much like a car does when it crashes into something. The object subjected to being shot or hit results in being damaged. Consequently, we can use the ontology to infer that whether you are shot or hit, this amounts to the same thing because either one will damage you. Step 5 shows a simple linkage between fire and damage through an equivalentTo relationship. Whilst they are syntactically distinct, the domain ontology shows that the term fire is linked to the term damage and crash is linked to the term damage via a causes relationship, which are linked to both shot and hit resulting in an equivalentTo relationship between shot and hit. If a car was projected into a shooting game then a car hitting something would be interpreted as being shot, whilst a gun fired in a racing game would be interpreted as a hit, because they both cause damage.

During the matching process a table is created containing the matched IOPEs from the behaviour request. The matched IOPEs act as keys in the table and have corresponding values, which represent the names of the IOPEs used in the behaviour ontologies. This process is important because the behaviour request and behaviour ontologies may refer to semantically equivalent IOPEs differently – the table of key-value pairs (stored in every environment capable of performing semantic interoperability) creates a semantic mapping between the different terms used. If all IOPEs in the behaviour request are matched this constitutes an abstract match and all ontologies associated with the Behaviour Profile (step 6), being retrieved. The ontologies are then returned to the environment (steps 7 and 8).

To determine whether the IOPEs defined in the behaviour request can be bound directly onto a signature provided by the target environment (a method with supporting input parameters and a return value), the Behaviour Profile is processed and the values associated with each IOPE are retrieved. These values specify which Atomic Process each IOPE belongs too in the Behaviour Process Model. The IOPEs may have been matched at an abstract level however they could belong to different atomic processes. Therefore, the framework tries to determine if a single atomic process exists that supports all of the IOPEs in the behaviour request. If so, the atomic processes, can be mapped onto the signatures defined in the list of behaviours supported by the target environment. Algorithms to achieve this are explained in more detail in [12]. If a match is found then the behaviours associated with the signatures are used to dynamically create the behaviours used by the object being projected into the target environment.

It is well documented that dynamically generating object behaviours is problematic. This can be attributed to the variation in how behaviour interfaces are defined and described, where one single difference in the ordering of parameters in the signature can render the behaviour inappropriate for generating dynamic behaviour. To accommodate this, mechanisms could be adopted similar to constructors used in object-oriented programming where a base constructor could be used and then extended to include the different ways the object can be created. Whilst this is one possible solution, it would be difficult to pre-define every possibility. A more effective way to address this limitation is to extend the concept of dynamic object generation of behaviours to enable signatures to be composed, resulting in new signatures emerging. We achieve this using intermediary behaviours and extended interfaces.

Using this approach, behaviours can be dynamically generated between objects either directly or indirectly through signature re-writing. It describes how signatures are constructed and indicates whether the intermediary behaviour itself can be directly invoked or whether it also requires intermediary services. This process allows environments to dynamically discover and generate behaviour conflicts that may occur and proactively establish compositions with intermediary services. This may result in several candidate objects that provide the same functionality.
Behaviours that best match the object capability requirements defined in the object request are added to an extended interface metadata object (an XML file that represents the re-written signature. It may comprise references to any number of different behaviours in order to produce a desired behaviour). The extended interface (EI) object is invoked when a behaviour provided by the object does not directly support a method invocation. This object behaviour has a fixed operation name called ‘EI’ which takes two parameters – the first is the extended interface metadata object and the second is an array containing all the parameters required to generate the behaviour at run time. This behaviour generation processes the extended interface metadata object, which provides information about the operation name for the intermediary behaviour, the parameters it takes, including the associated data type information, and the order in which the parameters appear in the signature.

If the connection mode is ‘direct’ the extended interface service uses the metadata for the intermediary behaviour to construct the required signature using the parameters in the array, before binding with it and executing the required method. In this instance, ‘direct’ means that object A can directly use a behaviour S1 provided by object B without having to use any intermediary behaviours. If the connection mode is ‘composite’ the extended interface behaviour generation processes the extended interface metadata object for the intermediary behaviour it needs before connecting to its extended interface service and passing it the metadata and the parameters. In this instance, ‘dynamic’ means that object A indirectly uses behaviour S1 provided by object B and S2 provided by object C. This process continues until a ‘direct’ connection with a required behaviour is established.

Using these techniques, behaviour interfaces can be evolved over time to accommodate behaviours they were not initially designed to process. For example, a car racing game that only implements a vehicle, which can crash and damage its surrounding objects in games other than those specifically designed to incorporate vehicles. The reader is referred to [12] for a more detail discussion on the algorithms used to achieve this.

B. Dynamically Generating Behaviours

The service ontologies map onto rule interfaces used to generate scripted behaviours dynamically. Our approach is based on the ReteOO algorithm [16], an implementation of the RETE algorithm, and utilises both database and scripting engines. This allows us to access information about objects and dynamically create object behaviours. As illustrated in Figure 2, metadata about objects, in this case a lamp is stored in a database.

Visual information is extracted from the database and, using the framework developed by Merabti et al. [12], transcoded if necessary. In parallel, semantic annotations describing the behaviours and functions the object provides are extracted from the database and added to the working memory of our rule engine (facts about behaviours are used to fire corresponding rules that provide implementation details about that behaviour. Figure 3 shows a partial sample query used to select this information about an object.

Data received from the database contains information, such as flag data, and links to more complex data structures such as XML serialisations for visualisation and semantic descriptions of behaviours. In this way adding an object to the virtual environment is achieved by inserting the required data into the database and object serialisations and semantic behaviour descriptions into pre-defined directories. Using this plug-in architecture allows us to easily add and remove objects dynamically from the environment.
Constructing the scripted behaviour is an iterative process which can result in many behaviours being added to the object. For example, a lamp could have a TurnOn, TurnOff, and Flash behaviour. This would result in three scripts being generated. One such rule to generate the TurnOn behaviour is shown in Figure 4.

Each line in the when part of the rule has a corresponding mapping to an atomic process in the Behaviour Process Model. This mapping allows high-level semantics (descriptions of what behaviours are supported by objects) linked to low-level scripted behaviours of the concepts used to describe any type of functionality.

Our initial findings show that to the best of our knowledge combining semantic annotations (behaviour ontologies) with rules to create dynamic scripting on the fly and project them into heterogeneous virtual environments is novel. We have moved a step further than other approaches, such as modding or world construction as in Second Life, to creating dynamic environments containing different artefacts with corresponding appearances and behaviours. This gives the user greater power to manage and use their content and functions independently to the constraints currently found in different virtual environments. In the next section we provide a brief overview of the technologies we have used to create our working prototype.

IV. IMPLEMENTATION

We used a peer-to-peer service-oriented framework developed to abstract functionality provided by networked devices and virtual environments, as services [Merabti et al.]. We have created corresponding digital representations of real devices, such as mobile phones, using Blender [19], which allows us to perform conversions between different 3D modelling formats such as COLLADA and OBJ. This functionality has been set up as a service in the framework developed by Merabti et al. [12].

We have utilised the Rhino Scripting engine provided by Java. Rules have also been implemented in Drools and we have developed our own virtual environments using the Java Monkey Engine and bridged various virtual environments and networked appliances with these environments to move, share and use content and functionality. One such virtual environment is illustrated in Figure 5.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a novel framework that describes how we project objects into heterogeneous environments through the interpretation of visualisation information and the generation of behaviours on the fly. We have used a database to store object information which is mapped onto plug-in data for objects (visualisation data and semantic descriptions of data). This information is pulled into our framework and used to create interpreted visualisation information and dynamic scripted behaviour from semantic descriptions to describe what functions the object provides. We exploit the concept of rules to understand these semantics and govern how behaviours are constructed in conformance with the scripting language supported by the destination environment and the behaviours it can support.

While the prototype has shown some early results that look very promising, this is still a work in progress where more comprehensive evaluations are required. The framework needs to be abstracted to represent efficiently many of the complexities it currently supports. For example, more flexible mechanisms that enable interoperability between different scripting languages including rule engines. Whilst we have in previous work shown that physical devices can be combined with corresponding artefacts within virtual environments, better case studies are required – we have begun to make progress on this in previous published works, and will continue in the future work.

REFERENCES


