

Java™: how Internet programming practically imagines virtuality

Adrian Mackenzie
Institute for Cultural Research,
Faculty of Social Sciences, Cartmel College
Lancaster University,
Bailrigg, LA14YR UK

Tel: (44) 01524 593380
Fax: (44) 01524 594273
email: a.mackenzie@lancaster.ac.uk

Abstract

The equation between the virtual and digital technologies that prevailed during much of the 1990s is now regarded as more or less untenable. This paper argues that another sense of the virtual remains intact and even vital in understanding the 'newness' of new media as cultural-technological processes. This notion of virtuality, loosely drawn from recent cultural theories of post-social relationality (Shields, Massumi, Knorr-Cetina), describes processes that involve divergent, ongoing, generative transformations in a given domain linking people and things. This paper analyses the notion of 'the virtual' at work in the production, circulation, and representation of the Internet programming language and software platform, Java. Examining the circulation, interpretations, coding practices, branding and implementation of Java, it suggests that a notion of *practical virtuality* can help understand the dynamism of new media as open-ended cultural-technical relationality.

1. Introduction: alternative concepts of the virtual

In the early-mid 1990s, new media seemed like 'spaces or places *apart from* the rest of social life ('real life' of offline life), spaces in which new forms of sociality were emerging, as well as bases for new identities, such as new relations to gender, 'race', or ontology' (Miller & Slater, 2000, 4). Examples of this separation of new media and ordinary life range across cyberpunk fiction, popular books (Rheingold, 1994), cinema (*Matrix*, *Strange Days*, etc) and much academic work on cyberspace. Theories of the virtual flourished as attempts were made to analyse the 'newness' of new media. Manuel Castells, for instance, wrote that virtuality is 'a system in which reality itself (that is, people's material/symbolic existence) is entirely captured, fully immersed in a virtual image setting, in the world of make believe, in which appearances are not just on the screen through

which experience is communicated, but they become the experience' (Castells, 2000, 373).

Today the principal objection to such notions of virtuality is precisely their tendency to separate experience of new media from another world, the real or ordinary world of social, cultural, economic and political activity. On the basis of that separation, a discontinuity between identities, bodies, relations, and ontologies associated with old and new media could be postulated as radically new. From the standpoint of 2003, post-dotcom crash and 9/11, the scope of virtuality and cyberspace seems much reduced. The identification of the Internet (including email, usenet groups, chatrooms, IRC, WWW, MUDS/MOOs, etc) with the virtual has become weaker and more unstable. In the main, notions of the virtual look like exaggerated representations of certain relational potentials of computer-mediated communication. As more recent studies of new media have shown, the identities, relations, politics and ontologies associated with new media are not radically different but intimately interconnected with older media, older institutions and older forms of sociality.

If the equation between new media and virtuality was mistaken or 'just' hype, what scope for an analysis of virtuality in relation to new media remains? Over the last decade or so, an alternative notion of the virtual has established itself and continues to extend its scope and relevance. The concept of virtuality associated with Gilles Deleuze's thought, and recently elaborated in (Shields, 2003; Grosz, 1999; Delanda, 2002; Massumi, 2002 in order of increasing difficulty) heads in a different direction. This version of the virtual designates something real that exists as a *multiplicity*.

As Rob Shields writes,

The virtual troubles any simple negation because it introduces multiplicity into the otherwise fixed category of the real. As such the tangible, actually real phenomenon ceases to be the sole, hegemonic examples of 'reality.' (Shields, 2003, 21)

Virtuality, as the mode of existence of a multiplicity, of something that continually generates divergences because it essentially consists of differences, seems to have some strongly relevant applications to a field as eventful as new media. The question is, can this alternate notion avoid separating between new media and some other more real domain? Can it resist the reduction of new media to a set of formal attributes (e.g. Manovich's principles of variability, modularity, transcoding, etc (Manovich, 2000))? Finally, can it enable analysis of the open-ended kinds of relationality associated with new media?

Researchers have been actively applying post-structuralist theory to new media (e.g. Lévy, 1998). But in his recent account of the virtual, Brian Massumi issues a warning about any direct identification of the virtual with new media:

Nothing is more destructive for the thinking and imaging of the virtual than equating it with the digital. All arts and technologies, as series of qualitative transformations ... envelop the virtual in one way or another. Digital technologies in fact have a remarkably weak connection to the virtual, by virtue of the

enormous power of their systemization of the possible. (Massumi, 2002, 137)

It probably still needs to be reiterated that new media and digital technologies are not actually or exclusively virtual. Some new media theory perhaps makes the identification too readily. The key issue here is avoiding an over-identification between new media and the virtual (in either sense), and yet remaining open to the forms of attachment, relationality and circulation associated with new media. Nevertheless, what are we to make of this warning? Does warning tell the whole story? Was the confusion between the virtual and the digital simply an error which over-valued digital technology's evident power to 'systemize' the possible? If it was an error, was it perhaps an interesting, deep and generative error? In short, do new media embody something generative, relational and open-ended in relation to collective life that this deeper, alternate concept of the virtual could help us analyse?

2. Practice and the error of virtuality

The Java™ Virtual Machine and the Java™ programming language provide an alternative way to read this involuted yet generative error. Dating from 1995, Java™ has gradually accreted more and more components, enrolled millions of programmers, been embedded in all the major web browsers, and triggered imitations and competitors (Microsoft Corporations .NET products). In 2003, it is being built into approximately 20 million mobile phones per month (according to some sources), comes pre-installed with every copy of Windows and MacOS, is found running on most web servers, and is distributed in various more esoteric forms such as smart cards. Java is the standard programming language taught to university computer science students. While it would be pointless to exhaustively list the applications of Java, the breadth and variety of its circulation suggests that Java™ embodies a new scaling up of technical mediation (Latour, 1996) within contemporary technological culture. Although technologies as objects or commodities have been blackboxed, branded and marketed for a long time, Java might be a salient instance where an object was successfully made into a practice and an interpretive community at the same time. Java as we will see is not so much a single thing, object or media, but an unfolding, bifurcating ensemble of practices, imaginings, logos, knowledges and artefacts loosely held together in various ways.

This alternate notion of the virtual seeks to provide a description of how things like Java become sites of attachment. Much of what we have so quickly come to take for granted about new media, the Internet, WWW and today, mobile devices, relies on the mundane acquisition and exercise of technical skills in programming and configuration of information technologies. While cinema, newspapers, magazines, science fiction novels, artworks, television, the Web, government reports, policies and legislation figure new media in many different ways, the production of new media

largely depends on technical practices. People work with specific hardware platforms, chosen computer languages and a narrow range of software applications. The production of new media requires the circulation of highly technical knowledges of many different kinds. These practices are highly context-specific.

They are nonetheless interlaced with cultural practices. Technical knowledge-practices overlap and enmesh with imaginings of sociality, individual identity, community, collectivity, organisation and enterprise. In offering a different concept of virtuality via Java, the aim is to understand how that overlapping-enmeshing process occurs. For instance, already in 1999, the Java-centred view of new media extended well beyond the desktop and screen:

By early in the new millennium, a large class of computational devices - from desktop machines to small appliances and portable devices- will be network-enabled. This trend not only impacts the way we use computers, but also changes the way we create applications for them: distributed applications are becoming the natural way to build software. "Distributed computing" is all about designing and building applications as a set of processes that are distributed across a network of machines and work together as an ensemble to solve a common problem (Freeman et al, 1999, 2)

The writer is a software engineer writing in a computer book entitled *JavaSpaces™*. The book is one amongst the thousands of trade press technical computer books that appeared during the 1990s (and continue to appear). These publications, especially the ones on software and programming, constitute a vast under-explored subterranean domain of new media and Internet culture. They can be regarded as a practical embodiment of the imaginings of virtuality that abounded during the 1990s. Many of these books open with a technological vision. As the example quoted above shows, the vision serves as a stimulus and motivation for the highly specific technological knowledge that follows. The proliferation of these often expensive yet short-lived books, situated somewhere between academic textbook, technical manual and extended advertisement, testify to the extraordinarily strong interest in practical knowledge of how to build or develop software during that time and today. In the late 1990s, being a Java programmer was not just to possess a very financially rewarding technical skill in programming. Much more than this, it represented a flexibility and identity for people and things closely tied to computer technology and new media.

3. Where is the virtual?

How can researchers into new media grasp the specificity of something like Java as a form of attachment and as an open-ended relationality? The historical context is relevant, since without it the specific nexus of forces and flows that give rise to Java make no sense. While Java could be situated within the history of the software industry (Campbell-Kelly, 2003) and the Internet (Abbate, 2000), its ongoing development and circulation makes it difficult to take a purely historical perspective. Too much is still in flux. In addition, without detailed archival work, any brief history of

Java risks repeating available quasi-journalistic accounts that appear in books such as (Lohr, 2002). Such accounts should themselves perhaps be part of the analysis rather than alternatives to it. More importantly, context alone does not necessarily bring the specificity of an object into view. It can just as well obscure that specificity. From the standpoint of important strands of recent science and technology studies (STS), the shifting ontological status of some objects, especially epistemic objects (Rheinberger, 1997) which are both the object of knowledge and technical work, complicates the researcher's work of contextualisation. In the process of their formation, socio-technical objects constantly contextualise and decontextualise themselves. Java is a typical 'variable ontology' object (Latour, 1996, 173) in that it continues even after almost 10 years to be in formation. It takes on different kinds of reality, it moves between social and technical registers, and it exists as an ongoing process of contextualisation and decontextualisation rather than simply as an inert object.

The prehistory of Java™ illustrates this variability. In the early 1990s, a software product group at Sun Microsystems in Palo Alto, California were working on Project Green. The software was designed to control all kinds of entertainment appliances such as video games and television set-top boxes and to allow them communicate with each other. Project Green had necessitated the design and implementation of a *platform neutral* programming language, initially named 'Oak' and later changed to 'Java' (Gosling, 1995, 1). In the Green project, the new programming language Oak aimed to overcome the obstacles that the particularities of commodity electronic hardware posed to programmers. When programmers were writing programs for mainframe computers, the computing environment could be quite well understood and well-defined. There were only a few mainframe manufacturers and even fewer operating systems. When programmers grappled with programmable consumer electronics, bugs multiplied. The new *lingua franca* of Oak was going to simplify the complexities of existing programming languages such as C and C++ and to implement a new platform for code which dispensed with some mundane but tricky aspects of programming embedded devices. The vision of many different consumer electronic devices with computers embedded in them coincides almost exactly with the current ideas of ubiquitous computing over a decade later (e.g. In *JavaSpaces* cited above). Like so many other projects, Green never came to market. Commercial agreements between Sun, set-top box manufacturers such as Mitsubishi and media companies such as Time-Warner fell through.

However, in the mid-1990s, at the same time as first-generation studies of new media and cyberspace were appearing, and the notion of virtuality as radically distinct from the real was most heavily discussed (Benedikt, 1991; Rheingold, 1994;) computer networks were becoming publically visible. The technical problem that Project Green had addressed was re-appearing in the guise of

differences between computing platforms. The motivation for Java, as told by one of the principal product engineers at Sun Microsystems, James Gosling, in a technical white paper from 1995 runs:

JAVA was designed to support applications on networks. In general, networks are composed of a variety of systems with a variety of CPU and operating system architectures. In order for an JAVA application to be able to execute anywhere on the network, the compiler generates an architecture neutral object file format – the compiled code is executable on many processors, given the presence of the JAVA runtime. (Gosling, 1995, 1)

A major practical claim is being made here. Java applications 'are able to execute anywhere on the network.' A scarcely veiled challenge to the practical existence of the computer program was implicit in this claim. If software applications can execute anywhere on the network, then they are freed or 'decontextualised' from the notoriously localised configurations and specificities of hardware. Much programming practice and system administration work tries to cope with the problems of platform specificity. An application written for Unix in the language C++ has to be re-written or 'ported' for Windows. Users of the latest version of Photoshop on Windows might have a slight different version of the program to those on Macintosh because different teams of programmers have trouble synchronising their work. A game written for the Sega Gamecube will not run on the Sony PlayStation. Many technical innovations and workarounds are used by software developers, system engineers and users to circumvent the obstacles that the differences between commodity hardware platforms poses. Many of these solutions involve re-writing or modifying software code at some level. Against all that, Sun trademarked the phrase *Write Once, Run Anywhere*[™] to highlight and brand the most potent promise of Java[™] applications.

4. The problem of the platform

New media rely on hardware and software platforms. Platforms tend to define themselves as 'lifted-out' spaces (Lash, 2002, 24) around which proprietary regimes, technological identities and consumption occur. Java effectively presented itself as a *meta-platform*, a space lifted-out from the already-lifted-out spaces of the Windows, MacOS or Unix platforms. The promise of a meta-platform was made to software developers rather than to the average consumer or user of hardware or software. Although Java made many headlines in mainstream newspapers and periodicals from 1995 to date, it has not been directly sold as a product to consumers. From the outset Sun Microsystems has freely *licensed* the basic tools and the Java Runtime Environment or Java Virtual Machine (<http://www.java.sun.com>), whilst retaining fairly strict control over the later evolution of the language. Issues of commodification, property and branding will return later, the key point about the 'Write once, run anywhere' slogan is that it promised something of practical significance to writers of computer software. The act of writing software, a complicated, highly labour-intensive activity, was literally 'disembedded' or detached from its existing context(s) in a key respect. The

program changed too. It became an 'architecture neutral object' rather an architecture specific one. The disembedding of programming work only occurred relative to the arrival of another platform, albeit of a different kind, the graphical web browser. Java's migration from consumer electronics to Internet programming language navigated a narrow, difficult passage via the web-browser. In 1995, the graphical web-browser, still relatively new at the time (Mosaic was released in 1993), could only display static text and static images. Many newspapers reported at the end of that year that Java was going to solve the 'problem' of static web content:

The most significant technical development of the year was Java, an 'object -oriented, interpreted, architecture-neutral programming language' according to its developer Sun Microsystems. That understated and unglamorous description hides the real power of Java, which will allow small programs - called applets - to be embedded in Web pages, giving them a life of their own (Azhar, 1995, 12).

If small Java programs or applets could be embedded in Web pages, then Web pages need longer be static. The web browser enhanced with Java could execute downloaded programs on the fly without any prior installation or configuration of those programs by a user. The condition of possibility of this capacity was that Netscape had agreed to implement and incorporate a Java Virtual Machine, a significant and sophisticated chunk of software, into Netscape Navigator, a proprietary product. Microsoft Corporation reluctantly followed soon after, licensing Java for Internet Explorer. It turns out that this promise of animating web pages did not really come to fruition. Many difficulties beset Java applets (speed of the virtual machine, download time, reliability, incompatible implementations of the virtual machine on different platforms, etc.). Today Java applets are relatively uncommon on web pages, even though they have technical capabilities that parallel those of the far more common Flash animations.

The promise to programmers of 'Write Once, Run Anywhere' took a different shape in the context of the struggle for domination of the browser market. In transmuting into a web browser plug-in, we could say that Java entered into an assemblage with the practices of web browsing and with the burgeoning domain of the World Wide Web. The proprietary differences between consumer electronic devices was no longer the problem it addressed. The technosocial problem of how to distribute the Java Virtual Machine to every computer connected to the Web was paramount. That distribution was entwined with the market struggle between Netscape Corporation and Microsoft Corporation, but also continually overflowed that struggle.

The history I have been re-telling so far has made two points. Following the lead of social studies of science and technology, cultural research into new media can understand its objects as possessing variable ontologies. Java initiated a variation in the mode of existence of computer programs and programmers in their relation to platforms. Platform specificity was henceforth no longer to be an

obstacle to the circulation of computer code. The new mode of existence of code was 'architecture neutral.' This in turn could be seen as changing programming work in certain ways. It became what I am calling for want of better term, *practically virtual*, that is, in some respects dis-embedded from proprietary platform specificities. Suspending existing proprietary platform specificities, Java found traction as a meta-platform through its linkage with a domain of intense cultural-technological interest. The advent of Java, which could have been just another of the innumerable series of software product releases coming from Silicon Valley, became significant in the market contest between Netscape and Microsoft. At the same time, Java's capacity to alter the mode of existence of programs and to change the character of the technical practices associated with programming required that initially at least it enter into assemblage with the highly contested domain of the graphical web browser.

5. Installing and implementing everywhere

A variation in the ontological status of code does not occur in isolation. Technical innovation occurs under mixed circumstances that bring together people and things, sometimes in inventive ways, sometimes in ways that hold things in place (Barry, 2001, 213). Two entities are brought together in Java. As mentioned previously, Java couples a programming language and a platform or system called the Java Virtual Machine. Two technical documents published as books formally define these entwined entities: *The Java Programming Language Specification* (Gosling, Joy & Steele, 1996) and the *Java Virtual Machine Specification* (Lindholm & Yellin, 1996). Although neither makes much sense without the other, they have different modes of existence, and they circulate in different ways. Therefore the sites where they are coupled or interact can trigger abrupt transitions or divergences.

For instance, the Java Virtual Machine (JVM) or the 'Java runtime,' as it usually known, was essential to Java's re-targeting for the Internet. As we have seen, Java's promise to programmers that they could write code once for different platforms relies on the Java runtime being installed everywhere the code is going to run. Installation presupposes a prior *implementation* of the virtual machine. The question of implementation has been a sore point in the installation of Java. In order to retain maximum platform neutrality, the *Java Virtual Machine Specification* remains highly formal in its description of the JVM. The specification states: '[t]his book specifies an abstract machine. It does not document any particular implementation of the Java Virtual Machine, including Sun's' (Lindholm & Yellin, 1996, Chapter 3). The motivation for this level of abstraction is simple. Different kinds of devices have different technical capacities: 'the Java Virtual machine does not assume any particular implementation technology or host platform ... It may also be implemented in microcode,

or directly in silicon' (Lindholm & Yellin, 1996, Ch 1). Many divergent implementations of the JVM arose in the late 1990s: Java-on-a-chip, JavaOS, JavaPC, PicoJava, Java smartcards, as well as the many implementations of Java for PC's, mainframes, workstations, and most notoriously, in the form of Microsoft's implementation which it licensed from Java. Microsoft's divergences from the abstract machine specified by in the *Java Virtual Machine Specification* led to ongoing litigation in U.S. courts between Sun and Microsoft. In both main cases, the installation of the JVM as a default component of the Windows operating was the core issue. In each case, Microsoft's decision to either include a non-standard implementation or no implementation succeeded, according to U.S. District Judge J. Frederick Motz, 'in creating an environment in which the distribution of Java on PCs is chaotic' (Olavsrud, 2002). This example shows that the coupling between Java the language and Java the platform was unstable and subject to divergence.

The coupling between programming language and installed virtual machine shows other instabilities and dynamisms. Some idea of mobility was important to nearly all visions of cyberspace and virtuality during the 1990s. While much attention has focused on analysing the experience of moving through infoscapes, and on understanding how that experience is intensified by the flows of information, not only information or content were becoming more mobile during the 1990s. As Java was implemented and installed, executable code was being mobilised. Not only information as content was moving , but code was beginning to move more rapidly. Whilst people did not treat the space of new media as a separate reality (Slater & Miller, 2000, 6), the reality of new media as practically instantiated in Java did entail separations or loosening of attachments between certain specific aspects of sociotechnical reality. Instead of programs being installed 'by hand', the Java Virtual Machine could in principle download code from anywhere on the Internet and execute it. The 'run anywhere' promised by Java was consonant with those discourses, but it required installation of the Java runtime at every point in the networks where Java code was to be run. At the same time as programming work in principle broke away from hardware specificities through the Java programming language, code or programs were being mobilised in ways that freed them from certain strictures to do with platform specificity.

A set of cultural processes of circulation involving both an object (JVM), artefacts linked to the object (programs or executable code) and the embodied skills of programmers now freed from some platform-specific constraints begin to coalesce around the abstract specifications of the language and the Java Virtual Machine. Together, these practicalities of programming work and mobilised code point to the circulation which the Internet has perhaps forcefully posed as a cultural problem.

6. Code as *pastiche* and *interpreted text: between books and machines*

Sifting through the avalanches of press releases, newspaper and magazine articles, on-line editorials and website discussions of Java over the last eight years gives some indication of the scope of the 'structured circulations' associated with Java. Put more abstractly, we could say that the variable ontology of a new media object such as Java is complicated by the structured circulation it undergoes.

Recent work in anthropology has begun to treat circulation as a phenomena in its own right:

[C]irculation is a cultural process with its own forms of abstraction, evaluation, and constraint, which are created by the interactions between specific types of circulating forms and the interpretive communities built around them. It is in these structured circulations that we identify cultures of circulation. (Lee & LiPuma, 2002, 192; see also Jones, 2002, TODO)

In these terms, we could regard the Java Virtual Machine and the Java programming languages as complex *circulating forms*. The abstract definition of form comes from the *Specifications*. The *Specification* exemplifies one form of abstraction and constraint on the JVM, a constraint that played an important role in legal struggles.

As the JVM was installed - unevenly and with incessant updates during the late 1990s - and the Java programming language become known more widely, an 'interpretive community' was growing up around these forms. There is something paradoxical about this, and it deepens the 'variable ontology' discussed above. Computer code or programs are often seen as the best exemplar of a text that can be read unambiguously. That is, programs seem exempt from divergent interpretations since they involve a strictly defined set of operations to be executed by some machine. How can there be an interpretive community or a communally constituted interpretive act in relation to a text that exemplifies strict self-identity? This question has wider implications for new media than just understanding the cultures of programming.

The space for an interpretive community resides in the gap between Java as a platform-neutral architecture or abstract machine and Java as a programming language. On the one hand, '[t]he Java Virtual Machine', writes the *Java Virtual Machine Specification*, 'knows nothing of the Java programming language, only of a particular file format, the `class` file format. A `class` file contains Java Virtual Machine instructions (or *bytecodes*) and a symbol table, as well as other ancillary information' (Lindholm & Yellin, 1996, Ch1: Introduction). Leaving aside technical terms such as 'class file' and 'bytecode', this quotation explicitly decouples the programming language Java from the JVM. The JVM 'knows nothing' of Java the language. On the other hand, '[t]he Java Virtual Machine,' the *Specification* later continues, 'was designed to support the Java programming language' (Lindholm & Yellin, 1996, Ch 2). The platform was designed to support a programming language it

'knows' nothing about. The relation between the programming language and the platform, even an architecture-neutral platform such as the abstractly defined JVM, seems contradictory.

One practical explanation for the contradiction lies in the difficulties of learning new programming languages. The syntax, lexical structure (the keywords, characters, operators, separators such as brackets) and expressions visible in Java programs were chosen to look very familiar to programmers. Explaining the motivation for a new programming language in 1995, the Java language designer James Gosling writes 'so even though we found that C++ was unsuitable, we tried to stick as close as possible to C++ in order to make the system more comprehensible' (Gosling, 1995, 2). While it was diverging from the economically and technically significant programming language C++ on key technical issues in order to fulfil the 'write once, run anywhere' promise, Java was also consciously evoking C++ through its syntax and lexical structures. We could say then that *pastiche*, the borrowing of elements from other writers or texts, explains how the JVM could both support the Java language and yet 'know nothing' about it. That is, the JVM serves a different purpose than Java as a programming language. The latter uses pastiche so that programmers find themselves working in a relatively familiar context. The former, the JVM, knows nothing of the language because it is only concerned with providing a mechanism that *isolates* executing code in various ways from the computing platform it runs on.

Pastiche is not unusual in programming language design. New languages often cite features from older, well-known programming languages so that the new language is easier to learn. Popular programming languages such as Perl pride themselves on borrowing the best features from many different languages (Wall, 2002). In order to understand how pastiche can be at the centre of an interpretive community, we need to go further into the practices of programming to understand how the JVM and Java the language interact as circulating forms. But the citational practices run deeper. Not only is the language syntax itself a pastiche of C++ and SmallTalk, but the practical writing of Java code nearly always relies on pastiche. The technical term for this is 'code reusability.' In relation to Java, perhaps the most significant symptom of this practice is the fact that the language is 'object-oriented'. The term is used in varying senses by software engineers and computer scientists, but in relation to Java programming, it refers to the fact that Java programs are structured around the citing or reuse of pieces of code ('objects') organised in a *class hierarchy*. Each class of objects inherits the attributes and behaviours of classes above it in the hierarchy and adds new variations to them. The work of programming in Java often consists in deciding whether a pre-defined class offers the desired behaviour, and then either invoking *instances* of that class in a program or *extending* that class by creating a modified version of it (Gosling, Joy & Steele, 1996, 128, 133). If no class offers the desired the behaviour, then a new type of class can be defined.

In the process, Java programmers spend much time trawling through 'the APIs', the Application Programmer Interfaces which define the pre-existing classes looking for desired behaviours in the class

hierarchy. The APIs are available in printed form (e.g. *The Java Almanac*; *Java in a Nutshell*, etc) or more widely as a large collection of online html files (e.g. <http://java.sun.com/j2se/1.4.1/docs/-api/>; June, 2003). Once programmers have learnt Java syntax, much of their programming work will revolve around reading APIs, and drawing together different pieces of the APIs to construct a more or less useful program. If there is a common interpreted text which underlies the production of Java code, it would include not only Java language syntax and idioms, but the class hierarchy documented in the API's as a central component. Over the last seven years, these API's have expanded significantly in response to the increasing differentiation and specialization of computer software. For instance, in 1998, a large class hierarchy named 'Swing' was incorporated into Java API (Sands, 1998). These classes contained highly configurable and carefully designed graphic user interface (GUI) components so that Java programs could be developed with the kinds of user interfaces seen in windows-style desktop computing software. The Swing components significantly increased the complexity of the Java APIs but at the same time provided a carefully designed contemporary 'look and feel' visual interface to the functionality of the Java Virtual Machine at a time when Java applets were losing their appeal. Like the many other additions to the Java API (SQL database connective, CORBA functionality, XML capabilities, etc), these components positioned Java in the rapidly changing domains of Internet and WWW innovation at the cost of increasingly fragmenting the technical knowledge-base of Java programmers. By 1999, it became difficult for any programmer to know all the of the Java APIs.

By virtue of both citational practices associated with Java (synactical pastiche and API code reuse), the identity of Java as a form starts to appear as only one provisional stabilisation of an interactive process of reading and writing which extends well beyond the frame of the onscreen code editor or development environment in which a programmer works. The APIs both stabilise and vary Java as a cultural new media text. The gap between the specific forms of the JVM and Java language functions not so much as a lacuna, but as a site of *metastability* where different social-technical problems are practically posed and receive divergent solutions.

7. Programmers as 'subjects of (new media) history'

Where and how does interpretation practically occur? Ethnographic studies of professional Java programmers at work observe them constantly shifting between onscreen and printed resources, and between onscreen windows (Mackenzie, 2003). One of the onscreen windows is likely to be open on an Integrated Development Environment (IDE), a piece of software that combines file management, code analysis, software building and other forms of programming assistance. Several dozen competing commercial and non-commercial IDEs (Borland JBuilder, IBM Visual Age, IntelliJ, Visual Cafe, VJ++, Sun One Studio, NetBeans, etc.) are used by professional and non-professional Java developers (Developer Tool Guide, 2003). An analysis of Integrated Development Environments used to organise and develop Java code would perhaps allow us to extend the interpretive community argument further. The organising

devices they offer augment the process of pastiche and citation in different ways. IDEs automate certain code development tasks so that programmers do not need to remember exact sequences of steps or look up the exact syntax of a particular code construct. For instance, 'code insight' devices automatically suggest possible completions of program code in the same way that 'auto-spell' features on word-processors suggest possible word completions. API documents are often available at a keystroke so that writing code and reading about existing reusable code become closely related operations. Often, these quite complicated artifices (code repositories, checkin-checkout mechanisms, automated build or 'make' tools) allow programmers to work in teams more easily. Problems of conflicting versions of code are managed by version control mechanisms and code repositories.

Beyond the mundane and highly repetitive sequences of action which IDE's seek to automate for programmers, the IDEs form part of the continuum shared by many other documents, website, journals, training programmes, product launches, trade shows, conferences and speciality cruises such as 'JavaJam' (Geek Cruises, 2003). These circulate practices, knowledges and visions concerning Java, but they also further *marketise* them. Software has been an industrial product for several decades (Campbell-Kelly, 2003), but arguably Java represents the first time in which coding work itself became an object of intense marketisation. There are a number of significant facets to this circulation that all cluster around the question of marketisation of programming itself.

Annual conferences such as 'JavaOne' held in San Francisco are heavily publicised within the software industry (JavaOne, 2003). The thousands of participants at these events (15000 in 2003) largely come from commercial or corporate IT industry and the computer trade press. Heavily funded by corporate sponsors such as Sun, these conferences combine product promotion, vision statements and training sessions in new or difficult parts of the Java API. Speculations on the future directions that Java will take, and how significant Java is to ongoing processes of technological innovation are legion. Vision statements typically suggest to Java programmers how significant their work is. At a recent conference, Sun's CEO Scott McNealy announced:

"The big message is that it is no longer cool to write for the operating system," said McNealy. "You don't write to Windows or Linux or Solaris or Macintosh. You write for the Java Web services layer. It is so last millennium to write for the operating system." (McNealy, 2003)

The direct claim here is that by coding in Java, Java programmers become 'the subject of history' or contemporary agents of innovation. Coding in Java, they inhabit the new millennium. These events position the programmers as agents of cultural and technical innovation.

Similar, albeit less grandiose, solicitations run through the many ancillary publications - print and on-line journals, magazines, books and websites - devoted to Java. For instance, the monthly on-line magazine *JavaWorld* has been publishing articles on Java since March 1996 (<http://www.javaworld.com>). These articles serve a mixture of purposes. Some are introductory tutorials written for beginning programmers.

Many of them publicise recent or forthcoming extensions of Java to new platforms or applications. Often, they focus on a specific facet of the Java language and demonstrate how it can be applied to a 'typical' programming situation. The definition of what counts as typical is highly fluid. A recent article bears the title 'A do-it-yourself framework for grid computing: learn to build your own grid computing application using open source tools' (Karre, 2003). Given that it is not likely that average Java programmers will be involved in implementing a 'grid computing application' (grid computing applications are large scale software installations involving many thousands of networked machines ranging from PCs to supercomputers; the most famous example is SETI, the Search for Extraterrestrial Intelligence, running on half a million PC users), why would *JavaWorld* be tutoring its readers in how to construct grid computing applications? Despite the fact that it massively exceeds the typical programmer's purview, grid computing is also seen as a way to take the Internet a step further. So the article suggests:

Driven by the success of the SETI project and others like it, researchers have been working to exploit the vast pool of computing resources connected to the Internet, but in a way that is secure, manageable, and extendable (Karre, 2003).

Many of these articles imply that only by reading and experimenting with the demonstration applications or code fragments described in the article, programmers become participants in the transformations 'about to' occur in the Internet. The 'vast pool of computing resources' in the quote above hints at these potentials.

8. Sites of attachment

Education in and marketisation of the skills involved in becoming a Java programmer have been intense and diverse. In the incessant updating of technical knowledge associated with the quite bewilderingly wide field of applications of Java, programmers and software developers become skilled along a number of different paths ranging across trade press, public institutions of higher education, hobbyist magazines and corporate education facilities. In the late 1990s, Java quickly became the teaching language of choice for introductory undergraduate computer science courses at universities across UK, Europe, USA, Australia and South-East Asia. Academic teaching departments were quick to respond to the appeal of Java to students as 'the' Internet programming language. Academic authors promptly produced programming textbooks based on Java (e.g. Rowe, 1997). These books merged into the stream of trade books flowing from publishers such as O'Reilly, SAMS, Addison-Wesley, etc). The academic and trade books are sometimes hard to tell apart.

At the same time as Java programmers were being trained in universities and colleges, the identity of 'Java Programmer' was being scaffolded by proprietary certification. For instance, the Sun Java Programmer Certification program administers tests of coding knowledge at Sylvan Prometric testing centres located in major cities around the globe (SunEd, 2003). At these testing centres,

programmers pay a substantial fee to have their knowledge of Java tested through a series of multiple choice questions downloaded via satellite from a central testing centre in the USA. If they answer at least 85% correctly, they receive a certificate, Java lapel badges and permission to use the Java logo on their business correspondence. These somewhat banal souvenirs carry *kudos* amongst some Java programmers. Achieving Java Certification tends to be the ambition of younger programmers who do not already have much industry or commercial experience, and whose employment prospects are still uncertain. In user groups, news lists and websites, debates intermittently arise over the real value of certification (see debates on [comp.lang.java.programmer](#) newsgroup over the last 5-6 years). The dispute is usually over the value of 'real experience' versus the book knowledge needed to pass tests. Underlying both sides of the debates, and through the debate themselves, a commitment to the distinct identity of the Java programmer is shared and reinforced.

If we stand back for a moment from the preceding discussion, and synoptically view what is happening with Java, what do we see? At the nucleus, stand relatively untouched, stable and inert textual forms such the *Java Language Specification* and *Java Virtual Machine Specification*. There is a gap between them. Although they relate to each other, that relation remains somewhat indeterminate and metastable. The trademarked promise 'Write Once, Run Anywhere' symbolically links these singular, relatively abstract sites of Java, the language and the virtual machine. They change at different rates and in different ways. As forms, they vary quite slowly in circulation. The Java language over the last seven years has only added small number of keywords. The JVM has changed little in specification.

At a second level of circulation, Java APIs and JVM implementations have diversified and proliferated massively. Certain aspects of Java became 'hotspots' attracting large amounts of programmer interest and coding work during the latter years of the 1990s. Sometimes, that interest becomes so intense that it led to changes in the form of Java or in some aspect of the work done with it. Soon after the difficulties with browser applets became apparent, new facets of the Java platform were emphasised. New sites of attachment or investment were made available for programmers. If applets were the 'front-end' application that elevated Java to semi-popular status via the graphical web browser, 'Java servlets' were the infrastructural software components that allowed increasingly complicated and dynamic institutional and corporate Internet websites to move away from adhoc CGI (Common Gateway Interface) Perl scripts that prevailed during the early 1990s. Like many other pieces of software infrastructure associated with growing popularity of the Internet and the advent of eCommerce, Java servlets and Java Server Pages (JSP) allowed websites to be integrated with existing software such as databases and transaction processing systems. They changed the nature of programmer's work by simplifying WWW programming through the code templates built into Java servlets. Java developers and software

architects easily found well-remunerated employment during the late 1990s partly because servlets offered organisations a more easily managed process of implementing complex database-driven websites. The character of the JVM implementations changed correspondingly. JVMs for specific purposes such as running servers or 'enterprise-wide' functionality appeared. The industry standard Java Enterprise Edition combined a number of different Java components into a package that could be used for 'enterprise-wide applications' (J2EE, 2003). Here, migrating inwards from the publically accessible side of corporate-institutional websites, Java percolated through the burgeoning and perhaps ultimately more economically important structures of business process reengineering and business-to-business (B2B) applications.

How do we explain the constant expansions, modifications and differentiations of the capacities and attributes of Java come from? It would be simplistic to attribute them to the creative energies of software engineers working at Sun Microsystems. Sun has been quick to copy innovations from elsewhere and incorporate them into Java, but it has often been following in the wake of other commercial, academic and community-group innovations. Java, as the programming language of the Internet (albeit in heavy competition with many other proprietary and non-proprietary languages), is both a way of structuring the circulation of new media things - through developing new applications and infrastructural frameworks - and a form that itself constantly absorbs innovations arising from its intensively marketised interpretive communities. The so-called 'Java Community Process', a scheme set up by Sun to elicit suggestions for the future development of Java, formalises this absorption.

In short, the relatively stable circulating forms of the Java language and virtual machines are heavily ornamented by layers of documents, manuals, tutorials which modulate that second layer of installation, implementation and code production. Many different pieces of software such as IDEs were designed to facilitate the writing of Java code. Beginning with graphical web browsers, crucial platform-specific implementations of the JVM were or are being made available by different vendors such as Sun, IBM, Apple, Microsoft, Redhat, etc. Around these relatively slowly evolving entities, a much more rapid and kaleidoscopic circulation of claims, counterclaims, promotions, conferences, press-releases, court cases, books, journals, and souvenirs associated with Java can be found. These repeatedly flow back through Java coding practices and the operational deployment of the Java Virtual Machine.

9. Conclusion: new media virtuality?

Where do we stand in relation to the alternate notion of the virtual discussed at the outset? The immediate equation between virtuality and digital technologies, as I said at outset, hardly seems tenable or relevant in 2003 except as an object of discursive analysis. However, through Java and the structured forms of circulation attached to it, a different kind of virtuality can be seen at play. One problem with the equation between digital and virtual was that it radically separated people

engaged in communication such as online communities (MUDs and MOOs) from their offline 'real' lives. In retrospect, it seems that very partial and relatively narrowly defined patterns of communication were read as totally transforming existing social relations and identities. Struggling to ground an explanation of shifting patterns of communication and sociality, many responses to new media premised their analysis on a radical disjunction between the virtual-digital and the actual-analog.

Whilst acknowledging the justification for this critique of the virtual, in exploring the practices of experimenting with, building, and maintaining forms such as Java, I would argue we see a different kind of virtuality unfolding. This virtuality is more subtle and open-ended than the one imagined by the often strident proclamations of the technological-virtual identity. It unfolds through technical yet deeply cultural processes of circulation associated with coding, networks and machines. Java begins as a typical variable ontology technosocial object, a programming language coupled to a chunk of software, the JVM. It successively bifurcates and differentiates as it enters into circulation. The pursuit of platform-neutrality exposes new sites of attachment, and enrolls new practices and processes of consumptive production. They modulate Java in turn. Because it remained metastable, Java accompanied and at times led the waves of innovation and change that visibly and invisibly changed information flows during the last decade. That is, it remained open to divergent realizations and it articulated diverse realities together.

The metastability resides in the relation between platform and coding language that Java contained from the outset. The problem of the platform underlies many debates around new media, ranging across questions of property, commodification, identity and social activity. The sociologist Scott Lash writes:

Platforms are particularly central forms of intellectual property. They are gatekeepers that bar and enable participation in in technological forms of life (Lash, 2002, 146)

Java promised to disembed software, a crucial kind of contemporary commodity, from the predominantly proprietary space of the operating system platform. At the same time, it offered programmers the opportunity to disembed themselves from the specificities of platforms that had for decades constricted their own already relatively elite employment and mobility. The disembedding of software from proprietary platforms and the release of programmers from the need to write for those proprietary platforms has proven to be powerfully attractive.

Looking beyond the field of new media, analyses of virtuality have something to offer wherever subject-centred or object-centred frameworks seem insufficient or decoupled, wherever meaning and materiality seem untenably separate.

'Today's tight connection of the virtual to the digital hardware and software is a new form. It represents

a return of 'the virtual' in our social activity. Some would just dismiss the term as an overused and underdefined label. However, this ironically recognises that, at a minimum, 'the virtual' is one of the most important marketing terms for the high-tech sector which is claimed to drive the development of a putative high-tech, knowledge-oriented 'virtual society.' (Shields, 2003, 19)

10. References

- Abbate, Janet. *Inventing the Internet* Cambridge, Mass. : MIT Press, 2000
- Azeem Azhar, 'Internet: net retains its capacity to surprise' THE GUARDIAN ONLINE *The Guardian* (London) December 28, 1995, 12
- Benedikt, Michael *Cyberspace: first steps* M.I.T. Press, Cambridge, Mass. 1991
- Campbell-Kelly, Martin *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (History of Computing Series), MIT Press, 2003
- Castells, Manuel *The rise of the network society*, Blackwell, Oxford, 2000
- Coombe, Rosemary J. *The cultural life of intellectual properties : authorship, appropriation and the law.* Durham, N.C. : Duke U.P., 1998
- Delanda, Manuel, *Intensive Science and Virtual Philosophy*, Continuum Books, London & New York, 2002
- Developer Tool Guide, *JavaWorld*, <http://www.javaworld.com/javaworld/tools/jw-tools-ide.html>, 25 June 2003
- Freeman, Eric, Hupfer, Susanne, Arnold, Ken, *JavaSpaces™ Principles, Patterns, and Practice*, Addison-Wesley, 1999
- Geek Cruises, 'JavaJam 4' http://www.geekcruises.com/home/jj4_home.html, 23 June 2003
- Gosling, James *Java: an Overview*, February 1995, [TODO - put URL]
- Gosling, James, Joy, Bill and Steele, Guy *Java Language Specification*, Addison-Wesley, Reading MA, 1996
- Grosz, Elizabeth, ed. 'Thinking the new: of futures yet unthought', *Becomings: Explorations in Time, Memory and Futures*, Cornell University Press, 1999
- Heiss, Janice, 'Scott McNealy at the 2003 JavaOne General Session' http://javaonline.mentorware.net/servlet/mware.servlets.StudentServlet?mt=1056383278531&mwaction=generic&subsysid=2000&file=janice_heiss3, 23 June 2003
- J2EE, 'Overview of the Java 2 Platform, Enterprise Edition (J2EE)' <http://java.sun.com/j2ee/overview.html>, (18 June 2003)
- JavaOne, <http://servlet.java.sun.com/javaone/> 23 June 2003
- Jones, Steve 'Music that moves: popular music, distribution and network technologies', *Cultural Studies* 16 (2) 2002 , 213 - 232

Karin Knorr Cetina & Urs Bruegger 'The Market as an Object of Attachment: Exploring Postsocial Relations in Financial Markets', *Canadian Journal of Sociology* 25, 2 (2000): 141-168.

Karre, Anthony, 'A do-it-yourself framework for grid computing: learn to build your own grid computing application using open source tools' <http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-grid.html>, (23 June 2003)

Lash, Scott *Critique of Information* Sage Publications, London 2002

Latour, Bruno *Aramis, or the love of technology*, trans. by Catherine Porter, Harvard U.P., 1996

Lévy, Pierre *Becoming Virtual: Reality in the Digital Age*, tr. Bononno, R. Plenum Press, New York 1998

Lindholm, Tim & Yellin, Frank *Java Virtual Machine Specification*, Addison-Wesley, Reading MA, 1996

Lohr, Stephen, Goto:

Manovich, Lev, *The language of new media* MIT Press, Cambridge, 2000

Massumi, Brian 'Too-blue: colour-patch for an expanded empiricism', *Cultural Studies* 14 (2) 2000, 177 - 226

Massumi, Brian, *Parables of the Virtual*, Duke University Press, Durham, N.C. 2002

Noble, James & Biddle, Robert, *Notes on Postmodern Programming*, Technical Report CS-TR-02/9, March 2002, School of Mathematical and Computing Science, Victoria University, Wellington, NZ

Olavsrud, Thor, 'Sun Wins Injunction Against Microsoft in Java Case'

<http://www.internetnews.com/ent-news/article.php/1561231>, December 24, 2002

Rheingenger, Hans-Jörg *Toward a History of Epistemic Things: Synthesizing Proteins in the Test Tube*, Stanford University Press, 1997

Rheingold, Howard *The virtual community: finding connection in a computerized world*. Secker & Warburg, 1994

Rowe, Glenn W. *An introduction to data structures and algorithms with Java*. Prentice Hall, London, 1997

Sands, John, 'JFC: An in-depth look at Sun's successor to AWT. Swing into great UI development' *JavaWorld*, January 1998, (<http://www.javaworld.com/javaworld/jw-01-1998/jw-01-jfc.html>; 12 June 2003)

Shields, Rob *The Virtual*, Routledge, London & New York, 2003

Sun Educational Services, <http://suned.sun.com/US/certification/java/index.html>, 19 June 2003

Ullman, Ellen, *Close to the Machine. Technophilia and its Discontents*, City Lights Books, San Francisco, 1997

Wall, Larry, 'Perl, the first postmodern computer language',
<http://www.perl.com/lpt/a/1999/03/pm.html> (3 July 2002)