Java™: reading, writing and running the Web

Adrian Mackenzie,
Institute for Cultural Research, Lancaster University, UK
a.mackenzie@lancaster.ac.uk

The key question this paper asks is: how can we understand the construction of the WWW as a process in time? The Web is the outcome of construction work on an unprecedented scale. It has grown rapidly, and changed its character in various ways.

In contrast to the many accounts of the Web focused on heroic inventors, formal attributes of the Web, or how the Web is used, the main material this paper uses to explore the dynamism of the Web is software, and particularly Java code. In his recent book on new media work in New York, Andrew Ross mentions that ‘philosophical clashes between technologists [programmers] and [web] designers had a long industrial history, stemming from differences in style, socialization, and trade vocabulary’ [Ross, 2004 #610, 66]. At core, there are disagreements over the nature of the material they are working with.

Programmers work with models and systems, designers work with visual materials, images and text.

The relation between programmers and web designers is an important part of the history of the WWW, not least because the Web was originally developed as a communication protocol rather than a design material. However, rather than seeing technology and design in conflict, this paper takes a different tack. It examines how the construction of the Web as designed visual-typographic-animated object and as processual information system are coupled in code. This code-centric view of the Web treats it as software, and regards software or code (Java in this case) as a primary material used in the development of the Web, by programmers and designers alike over the last decade. It focuses on code as cultural process rather than as technical function or infrastructure. The notion of technical function cannot possibly account for how code has changed over the last 10 years. Java, however, provides a particularly rich way to track some of the crossovers between programming and design.
As this website shows, Java tries to cover a lot of ground. It offers many interfaces ranging across commerce, popular culture, entertainment, education, science and technological innovation. It is both highly graphic object and extensively infrastructural. It is globally distributed and an object of attachment, something that people identify with. It is heavily branded and commodified, and yet its consumption overflows the branding in certain ways (for instance, in some important open source Java projects such as Apache Tomcat). In order to map the migration of Java code in the construction of the WWW, and to offer a more general theoretical model for some of the dynamism of that construction, this paper draws on three general concepts from contemporary social theory: (1) agency, the concept of who does what; (2) materiality, or what counts as the basic stuff that exists; and (3) sociality, the concept of how we belong together, how attachments form or collective social life coheres. Agency, materiality and sociality are high-level theoretical abstractions, but ones that criss-cross social and cultural debates over the politics, work, technology,
production and consumption of the Web.

**Agency 'under construction'**

Challenges to conventional notions of agency emerged in social theory over the last few decades. In general, they were motivated by the desire to understand why social and cultural changes were hard to predict or control. Their main target has been understandings of agency based on mental states such as intention. Intentions as the origin of social actions have been replaced by understandings of actions emerging from networks of relations between bodies and things. Occasionally, an intention does something, but that is an exceptional event. How does a relational view of agency change our understanding of the construction and design of the Web?

The Web has been represented and understood in terms of different attributes and properties, such as hypertextuality, virtuality, interactivity, digitality and more generally as 'new media' (see [Lister, 2003 #42] introductory chapter). It has been associated with processes of identity formation, new modes of production, commodification and consumption (digital economy), and as a re-invented public sphere. These conceptualisations of the WWW carry with them notions of agency, either in relation to what the Web is as a technology or what people do with the Web as they make-use it. In this split between Web as technology and Web as a product of people's actions, what place does code have? What does it do? Here, a code-centric view suggests a shift in perspective. In terms of code, we can somewhat abstractly but not correctly understand the Web as a vast, distributed software system or program. As this program runs, what does it do? What is its output? It's output is nothing other than a continually writing and re-writing of the program itself. There is no single end product or outcome of what the program does. The construction of the Web alternates between writing and running code. One way to track this alternation is through the way in which code was written for and run on the web.

From 1995 onwards, the Java™ Virtual Machine gathered more and more components, was embedded in all major web browsers, and triggered imitations and competitors (Microsoft Corporations .NET products). In 2003, it was being built into approximately 20
million mobile phones per month (according to some sources), came pre-installed with most copies of Windows and Mac OS, was found running on most web servers, and was distributed in various more esoteric forms such as smart cards. Java became the standard programming language taught to computer science students. As a programming language, it gained increasing popularity because changed the relation between writing and running code. It promised software developers they could write code that would run on different machines without being re-written. As we will see, Java code is symptomatic of important dimensions of the construction of the Web, if it not by any means the most common code material found on the Web.

**Practical virtuality**

At the core of Java as a species of code lies a concept of virtuality. Notions of the virtual and virtuality had great significance during the 1990s as a way of understanding large-scale social, economic and political change. There was much discussion of virtual societies, virtual identities, virtual spaces, virtual objects and so on. In most respects, the concept of virtuality has lost currency. It is no longer regarded as a good way of describing new media such as the Web. (Miller and Slater 2000) write in 2000: ‘[b]y focusing on “virtuality” as the defining feature of many Internet media and then moving on to notions such as “cyberspace”, we start from an *assumption* that it is opposed to and disembedded from the real’ (4). In the main, notions of the virtual look like exaggerated representations of certain relational potentials of computer-mediated communication.¹

Parallel to the ideological understandings of the virtual, a practical understanding of the virtual was emerging in Web programming. In this *practical virtuality*, disembedding and separating code from their contexts came to be viewed as desirable. Java was one such project that sought to de-contextualise code itself. That is, Java was intended to create a place apart from the ‘real world’ of programming as it existed in the software industries of the early 1990s. The virtual machine and programming language were the two components of the disembedding process, which, it was envisioned, would very extend beyond computer screens to many other computational platforms:

¹By 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, 1999 #33]

The ambiguities of the last sentence are relevant: not only was Java designed with a view to changes in information devices and networks that would go beyond Web pages on computer monitors, it was conceived as changing the way software is designed and built. The 'common problem', the problem to which the practical virtuality of Java responds, is how to make software amidst ensembles of machines.

In the mid-1990s, at the same time as the Web and first-generation accounts of new media and cyberspace were appearing [Rheingold, 1994 #47; Heim, 1993 #36], the technical problem that virtuality addressed appeared in the guise of disparities between computing platforms increasingly networked together by the Internet. 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 #58, 1]

Java applications 'are able to execute anywhere on the network.' A strong challenge to the prevailing mode of existence of the computer program was implicit in this claim of 'architecture neutrality'. 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. Many technical innovations and
workarounds are used by software developers, system engineers and users to get over the obstacles of differences between commodity hardware platforms. Many of these solutions involve re-writing or modifying software code at some level. These difficulties might have remained largely invisible to anyone outside the work of software development if Java had just offered a practical solution.

The popularity of scripting and interpreted languages such as Java, Perl, PHP, Python, Actionscript, and Javascript suggests that the promise of practical virtuality conveyed by Write Once, Run Anywhere, had wide appeal in the construction of the Web. When Sun Microsystems trademarked the phrase Write Once, Run Anywhere™, a more resonant promise was being made. This promise concerned the agency of code, what it could do. It modulated the relation between programmers' acts of writing and the program's act of execution. These two actions would coalesce or resonate rather than remain separate, provisional or contingent on hardware differences.

**Infrastructural fantasies – the meta-platform**

However, in order to virtualise itself, to move itself out of localised contexts, Java had to take on different kinds of reality. It had to move between social and technical registers, and participates in processes of contextualisation and decontextualisation rather than circulating as an inert, stable object. For instance, in 1996 Java was announced as a way of animating Web pages. Because virtual machines were to be embedded in Web-browsers such as Netscape Navigator, Java programs (applets) could be downloaded and executed anywhere that a Java virtual machine was connected to the Internet. The promise of Java applets was that they would animate static WWW pages. In particular, a vision of multi-media interactivity was at the forefront of the promotion of Java.

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 [Azeem, 1995 #56, 12].

If small Java programs or applets could be embedded in Web pages, then Web pages need longer be static. The Java-enhanced web browser could execute downloaded programs on the fly without any prior installation or configuration of those programs by a user. Netscape Corporation agreed to implement and incorporate a Java Virtual Machine, a substantial and sophisticated chunk of code, into Netscape Navigator, a proprietary product. Microsoft Corporation reluctantly followed soon after, licensing Java for Internet Explorer. It turns out that the promise of animated, audiovisual, interactive web pages did not really come to fruition through Java. Many difficulties beset Java applets (speed of the virtual machine in running graphics code, download time for Java applets, reliability, incompatible implementations of the virtual machine on different platforms, and complexity of programming applets in comparison to other techniques such as Flash, etc.). Today Java applets are relatively uncommon on web pages, even though they have technical capabilities that parallel those of Flash animations.

Nevertheless, 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 formed part of an assemblage that included the practices of webpage construction, and linking together of the World Wide Web. Java's general vision of code executing 'anywhere on the network' was no replaced by the promise of animating the World Wide Web. Arguably, linking to notions of interactivity, accessibility of information, hypertextuality, and re-distributions of authorship associated with the Web first gave Java credibility as a meta-platform, a singularly lifted-out space.

Agency, or who does what, is complicated by Java. The Web stages a complicated contest over agency. The power to act constantly shifts there. The ideals of hypertextuality and interactivity that guided much Web design are underpinned by the same desire expressed in Java's virtuality. But the promise of Write Once, Run Anywhere could never be fulfilled. In terms of Java, the Web is both the ground on which virtuality first gains traction and becomes practical, and also the place where code never executes fully, where there is
always more writing to be done, where the agency of the programmer is constantly tested against and struggles with the agency of the running program (too slow, slightly incompatible, too complicated, etc). In Java's practical virtuality, and perhaps in many other associated with the Web, agency partly remains a promise, something that will be done.

**Materiality and textual environments**

Although Web design structures information for retrieval, it also deals intensively with sensation and perception. In commonsense understandings, sensation and perception, let alone materiality, are not intrinsically political or social. They are just given, they are the stuff of experience and of the world. Much recent social theory has questioned this understanding of materiality as simply given. What counts as matter, and the outlines of what we touch, see, and hear, are now regarded as a social-historical zone of contestation. Perceptions of materiality, and the materialities of perception figure centrally in relation to the Web. Just as it can be seen as a contestation of agency, of who does what, the Web can also be seen as a re-materialization of existing media such as book, comic, catalogue, newspaper, magazine and album and existing habits of looking, reading, touching, and hearing.

As we have seen, Java initially promised to animate web pages by making them programmable. Many applets produced in the late 1990s for scientific, art, business and entertainment websites made heavy use of the graphics capabilities of the Java Virtual Machine and Java programming language. In the first five years of its circulation, the graphics and user interface libraries of Java were expanded massively to include sophisticated typographic controls, various graphic and image manipulation libraries, and cutting edge graphic user interface widgets. These even included HTML display components, so that web browsers could be built in Java software very easily.

For instance, in 1998, a large class hierarchy named 'Swing' was incorporated into Java API [Sands, 1998 #54]. This class 'library' contained highly configurable and carefully designed
How should the intense focus on graphics, animation and graphic user interface interactivity in Java be evaluated? Applets had already proved unpopular. It opposed the direction in which the architecture and design of the WWW was heading. The graphic capabilities of the JVM sought to mirror those of software written for desktop GUI applications. By trying to move these kinds of visual constructions and interactions into webpages, Java was swimming against the tide. More and more operational code was moving back onto servers, and layout, graphics and animation were being handed over to web browsers. While some significant desktop software was and continues to be written in Java, we could say that as a graphic programming language, Java failed to become widely visible. The idea that whole desktop-software style interactions would be appear in web pages as Java applets languished.

However, Java’s graphic user interface extravaganza perhaps served another purpose. It helped make Java code visible as the WWW was constructed.
new ways. The operational code for a website was cut into separate pieces. Instead of large single pieces of code, websites many rely on many small pieces of code, themselves often connected to other executing programs such as databases, other websites and other servers.
New facets of Java platform as architectural or infrastructural material were emphasized in the late 1990s. New sites of attachment became available to programmers. Although applets were the 'front-end' application that elevated Java to semi-popular status via the graphical web browser, the Java programming language and Java Virtual Machine were quickly expanded to include other bodies of code.

These libraries, some written and distributed by Sun Microsystems, and some produced
through commercial and non-commercial projects, extended Java’s reach into network infrastructures and protocols, databases, text-processing, encryption, internationalization, inter-system communication, and audiovisual data (sound and image). These extensions accompanied the ongoing materialization of the WWW as a platform of platforms.

For instance, Java servlets were the infrastructural software components that allowed increasingly complicated and dynamic institutional and corporate Internet websites to move away from ad hoc CGI (Common Gateway Interface) Perl scripts developed during the early 1990s. Like many other pieces of software infrastructure associated with growing popularity of the Internet and the advent of e-commerce, Java servlets and Java Server Pages (JSP) allowed websites to be meshed with existing information systems such as databases and transaction processing. They changed the nature of Web design work because HTML code was generated dynamically by code templates built into Java servlets. The composition of web pages changed from statically-displayed aggregates to dynamic mixtures of browser-side and server-side events. Java developers and software architects quickly found well-remunerated employment during the late 1990s partly because servlets offered organizations a manageable process of implementing complex database-driven websites. As websites became processually more complex, the character of 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’ [Sun Microsystems, 2003 #191]. These components were themselves incorporated into secondary products such as IBM Websphere or into open source projects such as Apache Foundation’s Jakarta Tomcat project. Migrating inwards from the publicly accessible side of corporate-institutional websites, Java percolated through the burgeoning and commercially important Web-based business process re-engineering and business-to-business (B2B) applications.

The withdrawal of Java as a graphic object, the growth of other Java API’s (Application Programmers Interfaces) and the proliferation of JVM’s reflects the ongoing materialization of the Web as object of perception, affect and cognitive work. Java and other code contributed to a gradually deepening and more intensive re-materialization of
text and images within processual composites. The growth of server-side code, and the increasing complementarities between Web design and information-architecture of the late 1990s, comes from the re-materialization of images, words and screen as sites of multi-level interaction. Through Java and other code, HTML-formatted text and image are woven intensively into trains of transaction, archiving, retrieval, sorting and searching. As Java code increasingly processed trains of events (transactions, searches, retrieval, messages, etc) on Web servers, it subsided into an information infrastructural material, where it plays an important role in the construction of process-driven websites. The withdrawal of Java as graphic material applies to the Web more generally as it becomes a general system of addressability (for instance, the embedding of web servers in devices such as cameras). Like Java, with its combination of intensive graphics and extensive infrastructural elements, the re-configured materiality of the Web plays out between visibility and invisibility, between what is perceived and experienced, and what largely invisibly operates to support perception.

**Sociality introjected into code**

The shifting materiality of the Web brings sociality, or the nature of collective belonging together, to the fore. Sociality is an abstract name for the relations that bond social groups and differences. Like other media, the Web is a form of sociality. But sociality, like materiality, is not given once and for all. A principle of uncertainty applies to social life: 'it’s a principle of uncertainty about what the collective is made of, or will be made of,' writes [Callon, 2002 #313, 288]. Sociality is mutable relationality, constantly challenged, mobilised and transformed. Collective is not given, but contested and subject to re-invention.
From this perspective, the primary object of Web design and programming work is sociality. As the Web expanded, one relation that came to the fore concerned who could access what information under what conditions. The general promise of the Web was access to all forms of information, anywhere, anytime. Yet there were also many projects seeking to regulate access to information. Web programming was central to the construction of increasingly complex barriers, thresholds and gateways on the web. Differential access could be used to generate economic, political and social value.

Apart from private ownership of internet infrastructure, code written in Java or other programming languages such as Perl, Javascript, Python, PHP, Lisp, and C++ was the primary means of constructing differential access to information and communication. Code permitted a wide variety of social relations to be 'introjected' [Virno, 2004 #395] or projected into the Web as primary site of information access in the last decade.

Java code for a simple graphic operation, drawing a circle, looks like this:

```java
import java.awt.*;
import javax.swing.*;

public class BasicDraw extends JComponent {
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        g2d.drawOval(0, 0, getSize().width-1, getSize().height-1);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.getContentPane().add(new BasicDraw());
        int frameWidth = 300;
        int frameHeight = 300;
        frame.setSize(frameWidth, frameHeight);
        frame.setVisible(true);
    }
}
```

While this code is implicitly organised by some social relations (the use of class would need to be discussed), it does not bring any of the interactions that the Web elicits from people in more complex communicative processes. However, Java code, as it subsided into Web infrastructure, took on a different texture doing the late 1990s. It became apparent
that the WWW differed from other media and forms of consumption because the everyday communication or 'interactions' of people visiting websites could be made more or less visible and processually manipulated through code. That is, as interactions occurring on the website (moving between pages, clicking on links, filling out forms, adding text) were channelled or captured through code running on the server, websites could channel communication and sociality itself in certain directions and not others. The dotcom boom was one symptom of that realization. That attempt to capture social relations in code structures gave rise to great uncertainty.

Java, like the Web, is a site of ambivalent relationality. On the one hand, practical virtuality allows code to move across different contexts and this is significant for programmers and for how the Web works as distributed software. On the other hand, the Web needs to leave open the possibility of not knowing who or what will be in communication. This is a contentious point. Not only does the web bring new kinds of sociality to the fore (for instance, who would have thought syndicating the diaries of miscellaneous individuals could be significant?), it also leaves open the possibility of ongoing changes in sociality. Perhaps all media do this, but the programmability of the Web, its existence as software, heightens the potential. An example of code indicates how potential communication is left open:

```java
protected XMultiComponentFactory getRemoteServiceManager(String unoUrl) throws java.lang.Exception {
  if (xRemoteContext == null) {
    // First step: create local component context, get local servicemanager and
    // ask it to create a UnoUrlResolver object with an XUnoUrlResolver interface
    XComponentContext xLocalContext =
      com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);
    XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();
    Object urlResolver = xLocalServiceManager.createInstanceWithContext(
        "com.sun.star.comp.helper.UnoUrlResolver", xLocalContext);
    XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
        XUnoUrlResolver.class, urlResolver);
    // Second step: use xUrlResolver interface to import the remote StarOffice.ServiceManager,
    // retrieve its property DefaultContext and get the remote servicemanager
```
While the code for the circle in a frame was representative of an important aspect of Java software (its relation to the generic graphic user interface of contemporary computers), it would be safe to say that most Java code for website interactivity looks something more this. The lumpy texture of this code comes from what is technically known as 'layers of indirection'. That is, the convoluted nature of this code attempts to provide ways of finding out where things are and what they do without anyone (human or machine) having to know exactly in advance where they are, or what they do. Most Java programmers, at least in North America and Europe, are working in organisationally-saturated environments. This context pervades the kind of code they are involved in reading and writing. In the rather unreadable but suggestive abundance of terms like 'Remote Context', 'createInstanceWithContext', 'Resolver', 'ServiceManager', 'MultiComponentFactory', 'queryInterface' and or 'UNO', 'Exception,' a contract is being developed that binds potential participants in communication. What is primarily at stake is an attempt to manage the possible variations in communication occurring in complex information environments. Uncertainty in the constitution of the collective is built into the code.

The Web has been a source of uncertainty in recent collectives. In it, sociality overflowed existing norms, conventions, laws, social functions and institutions. Reading code suggests one way of accounting for this uncertainty. Code is textured by relations between programmers, but it also captures relations in social contexts. The uncertainty comes from relations between relations.
Conclusion

The science fiction author Bruce Sterling described in a recent address to SIGGRAPH the emergence of ‘a picture of a new and different kind of physicality. It’s a new relationship between humans and objects’ [Sterling, 2004 #15]. In this ‘new and different kind of physicality’, materiality, agency and sociality are all at stake. As more things, places and events become web-addressable through http protocols, seeing, touching, and feeling change. As trains of events are increasingly percolated through code-structures, then doing something or communicating feels different.

If agency, materiality and sociality are all unstable in code, what does this mean for the construction of the Web? From the Java perspective, the history of Web construction is animated by instabilities of agency, materiality and sociality. As other platforms, places and space become associated with the Web, Java code has continued to diversify, migrate, and mutate. Identification with Java as Web programming language on the part of programmers feed the proliferation of Java as software infrastructures. Not so much a conflict or disagreement between technologists and designers occurs here. Rather the technology itself, that which is destined to submerge into infrastructure, has become an object of design and consumption, rendered aesthetic and branded as object of identification. In terms of agency, the oscillation between writing and running code means that actions, innovations and inventions are always relational events, tested and modified in the light of difficulties, obstacles, and promises of greater mobility. In terms of materiality, as we have seen, the interplay between what Java makes perceptible (shapes, movements, colours) and what remains less perceptible (transactions, algorithmic processes, trains of events) has been at the core of this continuous change. Finally, in terms of agency, the ‘lesson’ of Java in the construction of the Web might be that this linkage between what appears on the screen and what happens off-screen is deeply multi-dimensional. Relations between programmers intersect with complicated relations running between social fields of consumption and production.

References


Gosling, J. 1995. 'Java: an Overview'.


Sands, J. 1998. 'JFC: An in-depth look at Sun's successor to AWT. Swing into great UI development' *JavaWorld*.

Sterling, B. 2004. 'When Blobjects Rule the Earth' *SIGGRAPH*. Los Angeles, CA.

Sun Microsystems 2003. 'Overview of the Java 2 Platform, Enterprise Edition (J2EE)'.

As more recent studies of new media have shown, the identities, relations, politics and ontologies associated with new media are not radically different or disembedded but intimately interconnected with older media, older institutions, places, spaces and forms of sociality. Instead of insisting on the virtual, the simulated or the post-modern, we should start from the assumption that the Internet is no more transcendent than markets, nations, cultures, institutions, or identities.

In the early 1990s, a software product group at Sun Microsystems in Palo Alto, California were working on 'Project Green', later renamed Java. The software was designed to control entertainment appliances such as video games and television set-top boxes and to allow appliances to communicate with each other. Project Green had involved the design and implementation of a platform neutral programming language, initially named 'Oak' and later changed to 'Java' (Gosling, 1995 #58, 1). In the Green project, the new programming language Oak aimed to overcome the obstacles that the proprietary specificities of commodity electronic hardware put in the way of programmers. When programmers were writing programs for mainframe, mini or microcomputers, the computing environment could be quite well understood and well-defined according to widely shared standards. There were relatively few operating systems. When programmers began to grapple with the proliferating varieties of programmable devices appearing in consumer electronics, domestic white goods, communications and entertainment equipment, 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 computing platform which smoothed over some mundane but tricky pitfalls of programming embedded devices. The vision of many different consumer electronic devices with computers embedded in them coincides almost exactly with the ideas of ubiquitous computing over a decade later (for instance, in the JavaSpaces literature cited above, but also in the Mobile Java technologies) and finds current echoes as computer hardware manufacturers such as Intel and Dell move into consumer electronics.