State of the Art and Future Trends in Distributed Systems and Ubiquitous Computing

Friedemann Mattern
Department of Computer Science
Swiss Federal Institute of Technology (ETH) Zurich
CH-8092 Zurich, Switzerland
mattern@inf.ethz.ch
August 2000

Abstract: This report summarizes trends in communication paradigms for distributed systems, mentions established and new software infrastructures for distributed systems (such as CORBA and Jini), and gives an overview of mobile code and mobile agent principles. It then explains the vision of Ubiquitous Computing and future networked smart devices including techniques such as RFID tags (or "smart labels"). It also discusses issues in spontaneous networking, service discovery, and related concepts. The report quotes liberally from a number of online resources found on the Web.

Keywords: RPC, RMI, software bus, publish/subscribe, CORBA, Jini, mobile code, mobile agents, pervasive computing, smart devices, spontaneous networking, context awareness, smart labels, RFID tags, middleware, distributed systems, ubiquitous computing

Introduction

A distributed system consists of several computers that communicate over a network to coordinate the actions and processes of a common application. Distributed systems techniques have gained much interest in recent years due to the proliferation of the Web and other Internet-based systems and services.

Well-established techniques such as interprocess communication and remote invocation, naming services, cryptographic security, distributed file systems, data replication, and distributed transaction mechanisms, provide the run-time infrastructure supporting today's networked applications [1]. The predominant model is still the traditional client-server architecture. However, application development for distributed systems now relies more and more on middleware support through the use of software frameworks (e.g., CORBA) that provide higher-level abstractions such as distributed shared objects, and on services including secure communication, authentication, yellow pages, and persistent storage mechanisms.

In the near future, distributed application frameworks will support mobile code, multimedia data streams, user and device mobility, and spontaneous networking [1]. Scalability, quality of service, and robustness with respect to partial component failures will become key issues.

Clearly, a shift towards large scale systems has occurred in recent years: Not only the pure Internet with its basic protocols, but the higher-level World Wide Web is becoming a standard platform upon which some distributed applications are being realized. Here, the Internet (or an all encompassing intranet) and its resources are viewed as the global environment in which the computations take place. Consequently, high-level protocols and standards, such as XML, enter the focus of distributed system research while low-level issues (such as operating system peculiarities) become less important.

Rapidly evolving network and computer technology, coupled with the exponential growth of the services and information available on the Internet, will soon bring us to the point where hundreds of millions of people will have fast, pervasive access to a phenomenal amount of information, through desktop machines at work, school and home, through televisions, phones, pagers, and car dashboards, from anywhere and everywhere [7]. The challenge of distributed system technology is to provide flexible and reliable infrastructures for such large-scale systems that meet the demands of developers, users and service providers.

Looking further into the future, essential techniques of distributed systems will be incorporated into an emerging new area, called "Ubiquitous Computing" [13]. The vision of Ubiquitous Computing (or "pervasive computing", as it is sometimes called) is in some sense a projection of the Internet phenomenon and the mobile phone proliferation phenomenon we observe today into the future, envisioning billions of communicating smart devices forming a world-wide distributed system several orders of magnitude larger than today's Internet.

Trends in Communication Paradigms

There are many ways that application software components residing on different machines can communicate with one another over a network. One low-level technique is to directly use the call interfaces of the network layer, such as the socket mechanism, together with a custom communication protocol.

An already classical alternative that fits well with the client-server model, is Remote Procedure Call (RPC). In this model, a component acts as a client when it makes a request of another component; it acts as a server when it responds to a request from a client. RPC makes calling an external procedure that resides in a different network node almost as simple as calling a local procedure. Arguments and return values are automatically packaged in an architecture-neutral format sent between the local and remote procedures.

For each remote procedure, the underlying RPC framework needs a so-called stub procedure on the client side (which acts as a proxy for the remote procedure) and a similar object on the server side. The role of the stub is to take the parameters passed in a regular local procedure call and pass them to the RPC system (which must be resident on both the client and server's nodes) [21]. Behind the scenes, the RPC system cooperates with the stubs on both sides to transfer the arguments and return values over the network [22].

To ease the creation of stubs, RPC toolkits include special tools. The programmer provides details of an RPC call in a form of specifications encoded in an Interface Definition Language (IDL). An IDL compiler is used to generate the stubs automatically from the IDL specifications [18]. The stubs can then be included with clients and servers [21]. Such RPC frameworks together with their toolkits are precursors of more general middleware platforms like DCE or CORBA.

While RPC is reasonably well suited for the procedural programming paradigm, it is not directly applicable to the object-oriented programming style that has gained much popularity in recent years. Here, Remote Method Invocation (RMI) - a newer technique for Java-based systems - enters the scene.

RMI is similar to RPC, but integrates the distributed object model into the Java language in a natural way [20]. With RMI, it is not necessary to describe the methods of remote objects in a separate IDL file. Instead, RMI works right off existing objects, providing seamless integration [32]. Furthermore, remote objects can be passed as parameters in remote method calls, a feature that RPC systems usually do not have.

In RPC systems, client-side stub code must be generated and linked into a client before a remote procedure call can be done. RMI is more dynamic in that respect: Owing to the Java capability of transferring code, the stubs that are needed for an invocation can be downloaded at runtime (in architecture neutral bytecode format) from a remote location, for example directly from the server just before the remote method is actually invoked. Internally, RMI makes use of object serialization to transmit arbitrary object types over the network, and because downloaded code can potentially be harmful to the system, it uses a security manager to check it.

RMI therefore fits well with the general trend of distributed systems becoming more and more dynamic.

RPC and RMI are basically synchronous calls: the client is blocked while the call is processed by the server. Asynchronous variants that are based on multithreading are difficult to handle and are error prone. However, the trend in distributed systems goes towards asynchronous and reactive systems - systems that cannot wait indefinitely long for a synchronous call to terminate. Typical examples are user interface systems or real-time systems.

Such reactive and asynchronous systems are better served by a newer and more abstract communication paradigm based on events. Events are simple asynchronous messages, they are attractive because they represent an intuitive (although somewhat restricted) way of modeling that something happened that is potentially of interest to some other objects. Distributed infrastructures based on this paradigm are often called "publish and subscribe middleware" or simply "software bus" or "event channel".

The concept of such a software bus is quite simple: All clients (i.e., consumers of events) are connected to a shared medium called the "bus". They announce their interest in a certain topic (i.e., type of events) by subscribing to it. Objects or processes that want to send a message or an event to clients publish it under a certain topic to the bus. If the receiver’s topic matches the sender’s topic, the message is forwarded to the receiver.

From a software design point of view, the event paradigm offers the benefits of direct notification instead of busy-probing: an object tells the environment to inform it when something happens, and when something happens it reacts accordingly [19]. It thus eliminates the need for consumers to periodically poll for new events. Furthermore, the concept easily allows the use of so-called adapters that act as programmable middlemen in the event streams. They may for example multiplex, filter, or aggregate events.

However, although the publish/subscribe paradigm is attractive due to its conceptual simplicity and has the benefit to decouple objects in space and time (while at the same time abstracting from the network topology and the heterogeneity of the components), it requires a powerful brokering mechanism that takes care of event delivery. Furthermore, its use requires that the implicit or explicit semantics of the brokering mechanism is carefully analyzed - a priori it is not clear how fast events are delivered, whether events remain for a certain time on the bus (so that for example subscribers who miss anything due to system failure can catch up on missed events) and whether there are guarantees on the delivery order. It should be noted that the publish/subscribe paradigm is basically a multicast scheme, and it is well-known from distributed systems theory that broadcast semantics, in particular for dynamic environments, is a non-trivial issue.

The overall trend in communication paradigms seems to be clear: It goes from the simple procedural paradigm requiring relatively little system support, via the object-oriented method invocation principle, towards more abstract schemes for loosely coupled asynchronous systems that require complex run-time infrastructures. One can expect that infrastructures that support such abstract communication paradigms on the global Internet scale will be important in the future, probably being integrated with the routing and management schemes for general Internet-based messaging services. Scalability of such infrastructures is a real challenge, however.

Trends in Software Infrastructures and Middleware for Distributed Systems

Middleware and software infrastructures for distributed systems (such as DCE, CORBA or Jini) provide basic communication facilities to application components and handle issues such as platform heterogeneity that are due to differing hardware systems, operating systems, or programming languages. Furthermore, they provide a set of standard services that are typically needed by distributed applications such as directory service or cryptographic security.

An early middleware system was DCE, which is still being used in many large applications. Communication in DCE is based on RPC, and it essentially provides directory services, security services (based on the well-known Kerberos system), and a distributed file service.

One of the most widely used infrastructures for distributed systems today is CORBA, the Common Object Request Broker Architecture, which is supported by a large industry consortium. In contrast to DCE it is based on an object-oriented model. The first CORBA standard was introduced in 1991, and it has undergone continual and significant revisions ever since [22]. Part of the specification describes IDL (Interface Definition Language) that all CORBA implementations must support. IDL is based on C++ and is used by CORBA applications to define the externally visible parts of object methods that can be invoked by other objects. It has a similar function than the IDL of RPC toolkits.

The central component of a CORBA system is the so-called object request broker (ORB). The ORB provides a mechanism for transparently communicating client requests to target object implementations. It simplifies distributed programming by decoupling the client from the details of the method invocations: When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller [37].

In addition to the ORB, CORBA defines so-called object frameworks: object services (application independent services at the system level like naming, trading, or security services), common facilities (application independent services at the application level like a printing service), and domain facilities (services for specific application fields).

CORBA has gained much momentum in industry and research. Implementations of the standard are available from a large number of vendors and exist even as freeware. CORBA supports all important programming languages, is suited for almost every combination of hardware and operating system, and is now also being used to realize mission-critical applications in industries as diverse as health care, telecommunications, banking, and manufacturing.

While in the past almost all additions to the CORBA specification have been integrated over time by the vendors into their products, this will probably become increasingly difficult in the future. The new CORBA-3 specification is huge and includes the following major additions: Internet integration (CORBA through firewalls, URL addressing for CORBA objects), quality of service control (fault tolerance, real-time), and a component model similar to EJB (Enterprise Java Beans). It is questionable whether CORBA and in particular implementations of the standard can, in the long run, adopt all these and other foreseeable developments in the field of distributed systems.

Furthermore, CORBA was conceived for static distributed systems, requires considerable resources at run time, and uses the traditional client-server model as the basic metaphor. It is therefore not well suited for small devices, highly dynamic systems, and services that are spontaneously integrated into a federation. This, however, is a major trend in distributed systems, to which Jini and similar systems are better adapted.

Jini is an infrastructure that runs on top of Java and RMI to create a federation of devices and software components implementing services. Jini enables any device that can run a Java Virtual Machine to interoperate with others by offering and using services. A service is defined to be an entity that can be used by a person, a program, or another service [23]. Typical examples for services are printing a document or translating from one data format to another, but also functional hardware devices are considered to be services.

All services are granted as leases. Each service, when in use, is registered as being leased by another service. Leases are time-dependent and have to be renewed upon expiration. If the lease is not renewed, then Jini removes the service from the list of services offered.

A device or a service uses a standard mechanism to register with the federation. First, it polls the local network to locate a so-called lookup service. Then, it registers itself with the lookup service. The lookup service is equivalent to a bulletin board for all services in the federation. It can store not only pointers to the services, but also the code for these services or proxies representing the service, as well as defining service characteristics and attributes (e.g., a printer may specify whether it supports color printing) [24].

When a client wants to use a service offered to the community, it can download the required proxy object from the lookup service (after having located and contacted the lookup), including any code such as device drivers or user interfaces from the lookup service. This code mobility means that clients can take advantage of services without pre-installing or loading drivers. The downloaded proxy object can then be used locally to interact directly with the selected device or service, any device specific details are hidden by the proxy.

Jini and similar infrastructures (e.g., Universal Plug and Play or UPnP, which is a service discovery framework at a somewhat lower level than Jini) are thus well-suited for highly dynamic distributed systems where components may be mobile and typically form spontaneous networks, an important trend in distributed systems [6].

Trends in Mobility

People have an increasing desire for ubiquitous access to information, anywhere, anyplace, and anytime. For that, they need mobile and portable devices, but also adequate communication systems and software infrastructures.

Mobile devices in the form of portable telephones, pagers, and notebook computers are now commonplace. Technologies such as WAP, GSM, and in particular UMTS and similar so-called third generation cellular communication standards will soon give rise to new mobile devices that provide fast and immediate (i.e., "always connected") access to the Internet.

However, mobile devices are currently poorly integrated. One example is data synchronization: Since from the mobile worker perspective it is crucial that data (such as phone numbers or calendar information) remain consistent between the various devices, automatic synchronization is a necessity. Current synchronization software consists of proprietary products that only allows synchronizing between specific devices and applications. The trend here goes towards standards (such as SyncML, propagated by an industry consortium formed in February 2000) or more general synchronization middleware [5].

Another infrastructure problem is transparent roaming. Although protocols and systems such as mobile IP provide users the freedom to roam beyond their home subnet while consistently maintaining their home IP address, this is not as simple as roaming in cellular phone networks and has several drawbacks with respect to efficiency and scalability.

Not only people and computing devices can be mobile, but also program code. By mobile code one understands executable program code that moves from a source machine to a target machine where it is being executed. Mobile code may help to support user mobility: Personalized environments can follow a user between computers [41]. Platform independence of mobile code is usually achieved by using scripting languages for which interpreters are available on most systems, or by compiling into some platform-independent representation such as Java bytecode.

Mobile code is an important programming paradigm and opens up new possibilities for structuring distributed software systems in an open and dynamically changing environment. It can improve speed, flexibility, structure, or ability to handle disconnections and it is particularly well-suited if adaptability and flexibility are among the main application requirements. It has applications in many areas, such as mobile computing, active networks, network management, resource discovery, software dissemination and configuration, electronic commerce, and information harvesting [8].

Java applets are a prominent example of mobile code components. Applets are best known as small Java programs, embedded in a Web page, that could be executed within the Web browser. However, applets together with the ubiquitous availability of the Java Virtual Machine, Java’s class loading mechanism, its code serialization feature, and RMI make Java a full mobile code system where arbitrary code can be downloaded over the network and executed locally. Of course, security is a prime concern in that context.

A more elaborate form of mobile code, based on the "push principle" as opposed to the "pull principle" of mere code downloading, are mobile agents [8]. They consist of self-contained software processes which can autonomously migrate from one host to a different host during their execution. In contrast to simple mobile code systems, mobile agents have navigational autonomy, they decide on their own (based on their programmed strategy and the current state of the context) whether and when they want to migrate. While roaming the Internet or a proprietary intranet and visiting other machines, they do some useful work on behalf of their owners or originators.

The use of mobile agents is increasingly explored by an expanding industry, and first commercial systems (e.g., Aglets, Voyager, Concordia) are now available. Most of these systems are based on Java for the programming of agents.

Compared to traditional distributed computing schemes, mobile agents promise (at least in many cases) to cope more efficiently and elegantly with a dynamic, heterogeneous, and open environment which is characteristic for today’s Internet. Certainly, electronic commerce is one of the most attractive areas in that respect: a mobile agent may act (on behalf of a user or owner) as a seller, buyer, or trader of goods, services, and information. Accordingly, mobile agents may go on a shopping tour in the Internet - they may locate the best or cheapest offerings on WWW servers, and when equipped with a negotiation strategy (i.e., if they are "intelligent agents") they may even do business transactions on behalf of their owners [2].

Another general application domain is searching for information in the Internet or information retrieval in large remote databases when queries cannot be anticipated. Other uses of mobile agent technology include monitoring, remote diagnosis, groupware applications, and entertainment.

In general, mobile agents seem to be a promising technology for the emerging open Internet-based service market. They are well-suited for the personalization of services, and dynamic code installation by agents is an elegant means to extend the functionality of existing devices and systems. Agent technology therefore enables the rapid deployment of new and value-added services. Furthermore, mobile code and mobile agents are of interest for future Ubiquitous Computing applications, where small mobile devices may be "spontaneously" updated with new functionality or context-dependent program code.

Some important problems remain to be solved, however. The most important issues are probably security concerns: protecting hosts from malicious agents, but more crucially also protecting agents and agent-based applications from malicious hosts. The second issue is crucial for applications such as electronic commerce in an open world, but unfortunately it is difficult to tackle. The main point is that, as an agent traverses multiple hosts which are trusted to different degrees, its state can be changed by its temporary hosts in ways that adversely impact its functionality [42]. Transactional semantics for migration (i.e., "exactly-once migration" in case of communication failures), interoperability with other systems, coordination issues, and the management of large societies of mobile agents also still pose interesting challenges. Furthermore, a seamless integration of mobile agents into the WWW environment is crucial for the success of mobile agent technology.

Trends in Ubiquitous Computing

The vision of Ubiquitous Computing is grounded in the firm believe among the scientific community that Moore’s Law (i.e., the observation that the computer power available on a chip approximately doubles every eighteen months) will hold for at least another 15 years. This means that in the next few years microprocessors will become so small and inexpensive that they can be embedded in almost everything - not only electrical devices, cars, household appliances, toys, and tools, but also such mundane things like pencils (e.g., to digitize everything we draw) or clothes. All these devices will be interwoven and connected together by wireless networks. In fact, communication technology is expected to make further dramatic improvements, which means that eventually billions of tiny and mobile processors will occupy the environment and be incorporated into many objects of the physical world.

Progress in technologies for sensors (and thus the ability to sense the environment), together with the expected increase in processing power and memory, will render classical devices or everyday objects "smart" - they may adapt to the environment and provide useful services in addition to their original purpose. Most of these new emerging "smart devices" will be small and therefore highly mobile; some might even be wearable and be worn much as eyeglasses are worn today. They will be equipped with spontaneous network capabilities and thus have access to any information or provide access to any service "on the net". Connected together and exchanging appropriate information, they form powerful systems.

Portable and wireless Internet information appliances are already now a hot topic in the computer industry. Soon everything from laptops to palmtops to electronic books, from cars to telephones to pagers, will access Internet services to accomplish user tasks, even if users have no idea that such access is taking place [15]. It is clear that today’s mobile phones and PDAs, connected to the Internet, are only first precursors of completely new devices and services that will emerge. This will give rise to many interesting new applications and business opportunities.

There are many concerns and issues to consider also on the political, legal and social level. Privacy is of course a primary concern when devices or smart everyday objects can be localized and traced, and when various objects we use daily report their state and sensor information to other objects. Another issue is information overload. Internet users are already now overwhelmed by the sheer volume of available information, and the problem will get worse as the Internet grows and we enter the era of Ubiquitous Computing. Search engines, shopbots, portals, and email filtering are existing technologies that allow the user to reduce the torrent to a manageable stream, but these technologies are still quite limited [15]. New technologies and services are definitely required.

It should be clear that we move only gradually towards the ultimate vision of Ubiquitous Computing. Much progress in information science, computer science, and material science is necessary to render the vision economically feasible and to overcome current technological hurdles such as energy consumption.

However, it is also clear that Ubiquitous Computing will, in the long run, have dramatic economical effects: Many new services are possible that transform the huge amount of information gathered by the smart devices into value for the human user, and an entire industry may be set up to establish and run the infrastructure for the new networked information appliances.

Applications for Ubiquitous Computing will be found in areas where the Internet already plays an important role, such as mobile commerce, telematics, and entertainment, but without doubt many other traditional areas (e.g., health care, education) and newly emerging areas will benefit from Ubiquitous Computing technologies.

Trends in Smart Devices

A study by the International Data Group forecasts that by 2002, more than 50% of the devices connected to the Internet will not be PCs. They will be appliances and smart devices.

"Smart device" is a fancy name for a whole range of future devices that come in various shapes and sizes and are designed for various task-specific purposes. They all have in common that they are equipped with embedded microprocessors and are connected (usually by wireless means) to other smart devices or directly to the Internet. Some of these devices may be equipped with appropriate task-specific sensors. Others, usually called "information appliances", allow users to gain direct, simple, and secure access to both relevant information (e.g., daily news and email) and services [9] [25]. Their user interface is based on speech recognition, gesture recognition, or some other advanced natural input mode technology that is appropriate for their purpose, size, and shape. Multimodal human-computer interaction techniques also help to identify persons and thus to protect the access to the device. All these devices are so highly optimized to particular tasks that they blend into the world and require little technical knowledge on the part of their users - they should be as simple to use as calculators, telephones or toasters [25].

Conceivable are for example screens that can be affixed to walls, doors and desks. The displays can be configured to present information such as weather, traffic, stock quotes or sports scores extracted from the Web. Once configured, users can place these displays wherever they feel convenient. As humans, we are accustomed to looking in particular places for particular pieces of information [26]. This way, dynamic information becomes much easier to find and assimilate - a user might, for example, place today’s weather forecast on the wardrobe dor.

Smart toys is another appealing prospect. Compared to an ordinary toy, a networked toy has access to a huge world of information and can be more responsive to its owner and environment. For example, a toy that teaches spelling can access a large dictionary. It can also invoke a speech recognition process running in a remote computer to convert a child’s story into text. Or it will know the current weather and other global or local news and events. A toy (such as a smart teddy bear) may also act as a telecommunication device or even a telepresence device and serve as an avatar for the friends and family of the toy owner [27].

Of course, many other types of smart devices are conceivable. Wearable computing devices will be used to keep people informed, connected, and entertained. Just like the carriage clock of 300 years ago that subsequently became a pocket watch and then a wrist watch, personal electronic devices will become items that can be worn as clothing, jewelry, and accessories [34]. Wearable electronics may even become a new clothing concept: Textiles that are electrically conductive but also soft and warm to touch now exist. As a result, one can relatively easily move audio, data, and power around a garment. Conductive fibers can be integrated into woven materials, and conductive inks allow electrically active patterns to be printed directly onto fabrics.

One of the unique aspects of mobile devices is that they can have an awareness of the location where they are used. However, location-awareness is only one aspect of context-awareness as the encompassing concept, which describes the ability of a device or program to sense, react to, or adapt to the environment in which it is running. Context-awareness enables new applications based on the special nature of the context, for example interactive guide maps. However, it may also be exploited in determining the form of interaction supported and to modify interface behavior. For example, in a tourist guide, increasing text size in poor lighting conditions or, in a car system, limiting unimportant feedback during periods of rapid manoeuvring [10].

A dominant constraint for many information appliances will be their power consumption. If we assume a future where many task-specific devices exist instead of few general purpose machines, clearly users will not be interested in charging or replacing dozens of batteries. For mobile and wearable devices, however, only one of the devices (e.g., a mobile phone) that a user carries will need to communicate with wide-area networks. Other, more power thrifty personal devices may communicate over a link that only covers a person's immediate space [38].

Prototypes of such personal area networks already exist. IBM developed a technology that uses a tiny electrical current to transmit information through the body of a person [28]. In this way, a user's identification and other information can be transmitted from one person to another, or even to a variety of everyday objects such as cars, public telephones and ATMs. The bandwidth is relatively small, but more than enough to carry identification, or medical information [29].

Networked embedded processors, which form the heart of all smart devices, will become an important research and development field. New types of low-power processors will be developed specifically for networked embedded applications. Of primary interest are also advances in networking technology that could allow large numbers of embedded computers to be interconnected so they can share information and work together as part of a larger system. Reliability is crucial in embedded computing systems since such systems will increasingly control critical functions where safety is a factor (e.g., braking, navigation, driving) and, in some applications, may be given authority to take some actions with little or no human intervention. Ensuring the reliability of networked embedded computing systems could be difficult since large, interconnected information systems are notorious for becoming unstable. Such problems could be magnified by the presence of millions of interconnected embedded systems [14].

Progress in material science, chemistry and physics will eventually change the appearance of information appliances. For example, light emitting polymer (LEP) displays that are now available in first prototypes offer many processing advantages, including the possibility to make flexible large-area or curved displays capable of delivering high-resolution video-rate images at low power consumption, visible in daylight and with wide viewing angles [33].

Somewhat speculative are techniques known as "electronic ink" or "electronic paper". Although first prototypes exist, there is still a long way to go until paper can be replaced by electronic versions. However, the impact of this technology, once it is available, is significant: just imagine carrying your calendar and contact list on a foldable piece of electronic paper, or pulling out a large screen from a mobile phone or a tubular scroll containing the remaining electronics of a PC! Combined with small GPS receivers (Casio already markets a wristwatch that receives and processes data from the GPS satellites), maps that display their exact location ("you are here") are conceivable.

Trends in Spontaneous Networks

The so-called networked home not only simplifies the installation of security systems or energy management systems, but adds communications features to arbitrary home appliances and greatly amplifies their convenience. For that purpose, systems such as HAVi (Home Audio/Video Interoperability Architecture) are being developed by major consumer electronic vendors. With such a standardized middleware layer, interoperability between devices of different vendors is being achieved, which allows the combination of functions that have previously been separated. It implies a transition for home electronics products from largely stand-alone single purpose devices to cooperating elements in a system environment [23].

The cornerstone of the networked home is a bridge, a so-called "home gateway" or "residential gateway", between the internal home network and the public Internet. Ideally, this will be an open and vendor-neutral small server that makes Internet-connected services available to any household and gives all device manufacturers and service providers a common software interface to ensure interoperability. It may also simplify IP telephony because it functions much like a PBX [25].

The emergence of information appliances and new types of connectivity is spurring a new form of networking: unmanaged, dynamic networks of devices, especially mobile devices, that spontaneously and unpredictably join and leave the network. Underlying network technologies already exist, for example Bluetooth. (Bluetooth uses low-cost, short-range radio links to connect multiple portable devices; it is designed so that when two Bluetooth devices come close together, they automatically detect each other and establish a network connection.) Consumers will expect these ad hoc, peer to peer networks to automatically form within the home, in networked vehicles, in office buildings, and in various arbitrary environments [25].

In such environments, the ability to dynamically discover devices and services ("service discovery") is a key component for making these networks useful [25]. Service discovery protocols provide a standard method for applications, services, and devices to describe and to advertise their capabilities to other applications, services, and devices, and to discover their capabilities. Such protocols also enable them to search other entities for a particular capability, and to request and establish interoperable sessions with these devices to utilize those capabilities [39].

The most important technologies to date for service discovery are Jini (as described in a previous section), Salutation, Universal Plug and Play (UPnP) from Microsoft, E-speak from Hewlett-Packard, and the Service Location Protocol (SLP), which has been jointly developed by researchers both from academia and industry as a widely accepted and usable Internet standard for service discovery.

However, a crucial point is service mediation: matching requests for services with service descriptions. This is not as easy as it seems (e.g., does or should a request for a 300 dpi printer match a 600 dpi printer?), and designing a framework for service types is an especially difficult problem for open, dynamic networks, in which the communicating parties may have never encountered each other before, and cannot assume shared code or architectures [40]. Technologies such as XML may provide at least a syntactical basis for that problem. Another technology currently developed by the World Wide Web Consortium (W3C) is the Resource Description Framework (RDF) which provides interoperability between applications that exchange machine-understandable information. However, all these technologies do not address the underlying fundamental conceptual and semantical problem of service mediation, which remains an important open issue.

Trends in Smart Identification

One of the major problems in Ubiquitous Computing is the identification of objects [5]. For retail-based applications, barcode labels are typically used, which have a number of drawbacks however (such as the visibility of the barcode and the fact that it is a read-only item). So-called smart labels or RFID tags represent a newer and more interesting concept for identifying objects [11].

A smart label is a small, low-power microchip combined with an antenna, implanted in a physical object. Each label has a unique serial number and can contain other programmable information. The information contained in a label can be transmitted to a nearby reader device by a radio frequency (RF) signal [35]. Hence, smart labels are contactless, and require neither touch nor line of sight. In particular, they work through plastic, wood, and other materials [30].

Such identification tags can be battery powered or unpowered, and can be manufactured in a variety of sizes and configurations. The simplest give out a single pre-programmed number when placed in the vicinity of a reader [30]. It is also possible to store a limited amount of information on the tags. A more elaborate form of smart labels are contactless smart cards. In addition to storage memory, they contain a processor (with an operating system) and are able to process data (e.g., to perform cryptographic functions and thus disclose their state only to authorized entities.) Smart card technology is already well-established (smart cards have been used for many years as secure tokens for access control and authentication), but low-power requirements for contactless variants pose new challenges. Research is currently also underway to produce tags that report information about their physical environment, such as temperature or position [30].

Battery-powered labels, so-called active tags, can transmit their information over relatively far distances. Their disadvantages are that they have a limited operating life, and are more expensive than passive devices [36]. Passive tags, on the other hand, collect the energy to operate from the RF field emitted by a reader device, and therefore do not need a battery. They are less expensive (approx. $ 0.1 - 1.0), but can only be sensed at distances of up to a few meters. In some of these passive systems, the antenna is replaced by non-metallic, conductive ink. The substrate of the labels is usually paper or plastics, yielding a paper-thin and flexible label, which can be self-adhesive and be printed on. The labels are small enough to be laminated between layers of paper or plastic to produce low-cost, consumable items.

Smart labels are already used to identify packages for express parcel services, airline baggage, and valuable goods in retail. Industry estimates the smart label market to reach 1 billion items in 2003. Future smart labels equipped with sensors and more processing power will be capable of monitoring their environmental conditions, actively sending alerts or messages, and recording a history of status information, location, and measurement data.

Once most products carry an RFID tag (similar to today’s ubiquitous barcode labels), scenarios that go beyond the original purpose of those tags are conceivable. For example, an intelligent refrigerator may use labels attached to the bottles, which would be useful for minibars in hotel rooms. Even more intriguing are scenarios where prescriptions and drugs talk to a home medicine cabinet, allowing the cabinet to tell us which of those items should not be taken together, in order to avoid harmful interactions [31]. In a similar manner, packaged food could talk to the microwave, enabling the microwave to automatically follow the preparation instructions.

The underlying idea is that everyday objects can, in some sense, become smart by attaching RFID labels to them and by equipping the environment with sensors for those labels. The information on the label would then not be merely an identity, but a URL-like pointer to an infinite amount of information somewhere in the Internet. This means that each object could get an electronic identity in addition to its physical structure [43]. These electronic identities might then interact within a virtual world, independently from the physical world. If more and more of the physical world is becoming "smart", and both worlds were more closely linked, they would both be richer [44].

Conversations between inanimate objects may come close to the ultimate vision of Ubiquitous Computing and may provide us with an unparalleled level of convenience and productivity [31]. Much remains to be done, however, on the conceptual level and quite some infrastructure has to be implemented before this vision becomes true. The long-term consequences of such a world of things that virtually talk to each other are not yet clear, but the prospects are fascinating.

Literature

(Classification scale: A = general and introductory, B = for people with general technical competence, C = for experts)

[1] G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems - Concepts and Design (Third Edition), Addison-Wesley, ISBN 0-201-62433-8, August 2000 (B)
[2] S. Fünfrocken, F. Mattern: Mobile Agents as an Architectural Concept for Internet-based Distributed Applications - The WASP Project Approach, in: Steinmetz (ed.) Proc. KiVS'99, Springer-Verlag, pp. 32-43, 1999 (C)
[3] N. Gershenfeld: When Things Start to Think, Henry Holt & Company, ISBN 0805058745, 1999 (A)
[4a] H.W. Gellersen (Ed.): Proc. Handheld and Ubiquitous Computing, Springer-Verlag, ISBN 3-540-66550-1, 1999 (C)
[4b] P. Thomas, H.W. Gellersen (Ed.): Proc. Second Int. Symp. Handheld and Ubiquitous Computing, Springer-Verlag, ISBN 3-540-41093-7, 2000 (C)
[5] U. Hansmann, L. Merk, M.S. Nicklous, T. Stober: Pervasive Computing Handbook, Springer-Verlag, ISBN 3-540-67122-6, Oct. 2000 (B)
[6] R. Kehr: Spontane Vernetzung, Informatik-Spektrum, Volume 23, Issue 3, pp. 161-172, 2000 (C)
[7] D. Kotz, R. Gray: Mobile Code - the Future of the Internet, Third International Conference on Autonomous Agents, Seattle, 1999 (B)
[8] D. Kotz, F. Mattern (Eds.): Agent Systems, Mobile Agents, and Applications, Springer-Verlag, ISBN 3-540-41052-X, 2000 (C)
[9] D.A. Norman: The Invisible Computer, MIT Press, ISBN 0262140659, 1998 (A)
[10] T. Rodden, K. Chervest, N. Davies, A. Dix: Exploiting Context in HCI Design for Mobile Systems, First Workshop on HCI for Mobile Devices, 1998 (C)
[11] K. Finkenzeller: RFID-Handbook, Wiley, ISBN 0-471-98851-0, 1999 (B)
[12] M. Weber: Verteilte Systeme, Spektrum-Verlag, ISBN 3-8274-0221-2, 1998 (B)
[13] M. Weiser: The Computer for the 21st Century, Scientific American, September 1991, pp. 94-10 (A)
[14] National Research Council: Study Committee on Networked Embedded Computers (B)

Quotations and text fragments also come from several Web resource. Although not all of them could be identified in retrospect, the most important are:

[15] www.cs.dartmouth.edu/~dfk/papers/kotz:future/
[16] www.cdk3.net/*
[17] lasim.univ-lyon1.fr/c.ray/bks/java/htm/ch17.htm
[18] www.sce.carleton.ca/courses/94587/Introduction-to-open-distributed-computing.html

[19] www.cs.caltech.edu/~adam/phd/why-events.html
[20] tns-www.lcs.mit.edu/manuals/java-rmi-alpha2/rmi-spec/rmi-intro.doc.html
[21] www.sce.carleton.ca/courses/94580/Introduction-to-open-distributed-computing.html
[22] www.cs.washington.edu/homes/abj3938/cse588/MotivatingCORBA.htm
[23] www.kom.e-technik.tu-darmstadt.de/acmmm99/ep/montvay/
[24] www.sun.com/jini/factsheet/
[25] www-3.ibm.com/pvc/ in particular:
http://www-3.ibm.com/pvc/index_noflash.shtml
http://www-3.ibm.com/pvc/sitemap.shtml
http://www-3.ibm.com/pvc/tech/salutation.shtml
http://www-3.ibm.com/pvc/pervasive.shtml
http://www-3.ibm.com/pvc/nethome/
[26] www.media.mit.edu/pia/Research/AnchoredDisplays/
[27] www.media.mit.edu/~r/academics/PhD/Generals/Hawley.html
[28] www.research.ibm.com/journal/sj/384/zimmerman.html
[29] www.research.ibm.com/resources/magazine/1997/issue_1/pan197.html
[30] www.media.mit.edu/ci/research/whitepaper/ci13.htm
[31] auto-id.mit.edu/index2.html
[32] docs.rinet.ru/JSol/ch16.htm
[33] http://www.cdtltd.co.uk/
[34] http://www.extra.research.philips.com/password/passw3_4.pdf
[35] http://www.aimglobal.org/technologies/rfid/resources/papers/rfid_basics_primer.htm
[36] http://www.cwt.vt.edu/faq/rfid.htm
[37] www.cs.vu.nl/~eliens/online/tutorials/corba/overview.html
[38] http://portolano.cs.washington.edu/proposal/
[39] http://www.eet.com/story/OEG20000110S0027
[40] http://www.ncsa.uiuc.edu/People/mcgrath/Discovery/dp.html
[41] http://www.cl.cam.ac.uk/users/dah28/thesis/thesis.html
[42] http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper033/SWARUP96.PDF
[43] http://www.media.mit.edu/~jofish/ieee.paper/ieee.cga.jofish.htm
[44] http://www.cooltown.hp.com/papers/webpres/WebPresence.htm