A Survey on Operating System Support for Embedded Systems Properties[1]

Luís Fernando Friedrich

Departamento de Informática e Estatística – Programa de Pós-Graduação em Ciência da Computação - Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brazil

fernando@inf.ufsc.br

Abstract. *We can find a large variety of applications where embedded systems play an important role, from small stand-alone systems, like a network router, to complex distributed real-time embedded systems (DRE) supporting several large scale mission-critical domains as avionic applications. The rapid progress in processor and sensor technology combined with the expanding diversity of application fields is placing enormous demands on the facilities that software infrastructure like operating systems must provide. The paper presents an examination of how some of the important properties of embedded systems has being supported by operating system infrastructure.*

1. Introduction

Embedded systems are becoming increasingly complex. The use of processor-based devices has increased dramatically for most of our activities, both professional and leisure. Nowadays, more and more embedded systems are implemented in a distributed way, a wide range of high-performance distributed embedded systems have been designed and deployed. As a lot of aspects of embedded system design become increasingly dependent on the effective interaction of distributed processors, it is clear that as much effort needs to be focused on software infrastructure, such as operating systems, in order to provide the properties (specially non-functional) they need.

This paper presents a list of important properties that are supposed to be supported at the current state of embedded systems, from small stand-alone to distributed real-time, and examine operating system infrastructure actually used to provide functionality for embedded systems. First we list some of the characteristics of embedded systems and present a classification. Then, it is presented some properties that usually are very useful for embedded systems. Next, we present operating system infrastructure alternatives which are intended to provide the necessary functional and non-functional requirements for embedded system software to execute. They were analyzed, based on their description, regarding how they support the properties presented. Finally, we present some conclusions on how OS infrastructure is supporting important properties for embedded systems.

2. Characterization of Embedded Systems

An embedded system is hidden inside a system or environment, performing some dedicated function. The word embedded indicates that the system or device is part of another (larger) system. The hosting system may be a specific system such as a car or an

1 This work was supported by CAPES / MEC – Brazil through Project BEX3342/08-5.

aircraft, a machine or a factory, but it may also be a person in the case of an intelligent pace maker or some hearing device, where the embedded system replaces or extends the human capabilities. Embedding computation into the environment and everyday objects (also called pervasive computing) would enable people to cooperate with information-processing devices in a more informal way than they currently do, and independent of their location or situation they find themselves.

As there is an enormous variety of embedded systems from small intelligent sensors to vast aircraft control systems and vehicle simulators, the functions they performed may vary a lot. Different embedded systems will need different functions, not all functions are necessary on an embedded system. For instance, a robot will not always have a human interface. Examples of how real-time and embedded systems provide us with services are in *Automotive* or *Avionics*, *Health and Medical Equipment*, *Consumer Electronics and Intelligent Homes*, and *Telecommunications*.

Unlike PCs and workstations that execute regular non-real-time general purpose applications, such as our editor and network browser, the computers and networks that run embedded real-time applications are often hidden from our view.

Actually, embedded systems are heading more and more towards networked. Rapid advances in microelectronic technology coupled with integration of microelectronic radios on the same board or even on the same chip has been a powerful driver of the proliferation of a new kind of Networked Embedded Systems (NES) over the last decade.

Concurrently, embedded systems are becoming smaller and smaller. In reality, while previously sensors were directly connected to the central computing elements (mostly in an analogue way), today, they are becoming embedded systems themselves. Sensors have a processor included and do some preprocessing of the measured physical property (like temperature, displacement, pressure, etc.) sending the results of this preprocessing to a central management subsystem via a digital network.

We are not intended to establish a new embedded systems taxonomy, but in order to be coherent our philosophy is one of defining embedded systems which in turn can be subdivided in 2 categories: Stand-alone embedded systems, networked embedded systems.

2.1. Stand alone embedded systems(SES)

Work in a stand alone mode taking input and producing the desired output. The last two decades have witnessed a significant evolution of stand alone embedded systems. These systems are now being assembled from Intellectual Property (IP) components which are assembled on a System-on-Chip (SoC). SoCs offer a potential for embedding complex functionalities, and to meet demanding performance requirements of applications such as DSPs, network, and multimedia processors.

Most of Consumer Electronics (CE) devices are classified as SES. For instance, the explosion of the CE market over the past decade has generated products mainly in three categories: *Low-end* devices generally are built around application specific hardware like ASICs or SoCs with small amounts of program memory (ROM), usually around 256 kbytes, and inexpensive processors. *Mid-range* consumer devices, such as video cameras, are characterized by moderate amounts of program memory like 1 to 2 Mbytes. *High-end* devices, such as smart phones and set-top boxes, usually have much more memory, up to 32 Mbytes. In most cases, they use powerful processors and are developed by large programming teams.

2.2. Networked embedded systems(NES)

Networked embedded systems may come in many different forms. These systems have been variously referred as EmNets (Network Systems of Embedded Computers) [NRC 2001], NEST (Networked Embedded System Technology) [DARPA IXO 2002], and DRE (Distributed Real-Time Embedded) [GENI 2006]. Fundamentally, networked embedded system is a collection of spatially and functionally distributed embedded nodes interconnected by means of wireline or wireless communication infrastructure and protocols, with some sensing and actuation elements interacting with the environment [NRC 2001].

In order to be in accordance to the different forms that Networked Embedded Systems may come, in this paper we consider tree types of networked embedded systems, based on what is proposed in [FP6-IP-RUNES 2005]: Embedded Systems, Sensor Systems, and Distributed Real-Time and Embedded.

Embedded Systems are systems where the computing components are embedded into some other purpose built device (an aircraft, a car, or a home). Here the characteristic is that these systems are usually not mobile and often not all devices are connected, usually with only one other server machine and most of the time not to external networks. The type of the connection is often wired.

Sensor Systems are most of the time composed by a large number of possibly tiny devices having a single task which is monitoring some conditions within an environment and report back to a central server. The most widespread sensor networks are usually not mobile but the sensors are connected through a wireless network. Wireless sensor networks are a widely deployed example of networked embedded systems. There is a great interest from both the industry and academia in wireless sensor networks technologies that enable deployment of a wide range of applications, such as military, environmental monitoring, e-health applications, etc.

Distributed real-time and embedded systems play an increasingly important role in modern application domains, including military command and control, avionics and air traffic control, and medicine and emergency response. Distributed real-time and embedded (DRE) systems outline a computational infrastructures of many large scale mission-critical domains where are used to control a variety of artifacts across a number of sites. In life-critical military DRE systems, such as those defending ships against missile attacks or controlling unmanned combat air vehicles through wireless links [Lee, Leung and Son (2008)], to provide the right answer at the right time is crucial.

3. Requirements of Embedded Systems

In software engineering, functional requirements specify specific behavior or functions of a software system or its component. In general, functional requirements define what a system is supposed to *do*. They are supported by non-functional requirements, which specify criteria that can be used to judge the operation of a system, rather than specific behaviors. In general, non-functional requirements define how a system is supposed to *be*. Non-functional requirements are often called qualities of a system.

Traditional, embedded software can be quite complex and have a number of requirements. These have implications both for the application and for the software infrastructure, such as the operating system. According to [Crespo, Ripoll, González-Harbour, and Lipari 2008], and Computer Science and Telecommunications Board [NRC 2001] embedded software have several common features such as the following:

(i) *Resource-constrained computing*. They are frequently rigorously constrained regarding available resources. As a result of these restrictions, the system needs to efficiently use its computational resources. For instance, the operating system must be able to operate in resource-constrained environments.

(ii) *Real-time requirements*. Because many embedded applications interact deeply with the real world, they often have strict real-time requirements. These applications require functionalities such as process control, multimedia processing, instrumentation, and so on, where the system has to fulfill a temporal requirement, or deadline.

(iii) *Portability*. Many different types of CPUs, peripheral chips, and memory architectures may be used in embedded systems. Thus, for low cost, any embedded OS or other reusable component that is meant to be used on multiple applications should be commonly portable to custom hardware platforms.

(iv) *High reliability*. Embedded systems are deployed remotely, often in infrastructure-critical applications. Software faults are thus very problematic and are extremely expensive or even impossible to fix.

Stand alone (SES) high-end devices usually have a high degree of human interaction, use relatively expensive high-end processors, and frequently also co-processors, that deliver great throughput. For that reason, the real-time requirements are not particularly demanding. In contrast, SES lower-end devices in general have relatively little human interaction and as an alternative consist largely of processes whose timing needs to be tightly controlled. For example, pressing the Digital Camera(DC) shutter starts a series of threads that might involve various tasks that must be completed before a calculated deadline. High-end devices often have special-purpose hardware, such as the Digital Signal Processors (DSP) used in Set Top Boxes (STB) in order to do demanding processing tasks such as IP routing and video decoding, removing the responsibility for that tasks from the OS, In contrast, lower-end devices do not allow having dedicated hardware in order to complete these tasks, so they must rely on the Operating System real-time performance.

As embedded systems get networked like NES, the scenario gets a little more complex and while some common requirements become more stringent other new requirements also become very important to provide. Now, support for many of the important attributes for embedded, sensor and distributed real-time embedded systems are extremely important. Such attributes include requirements that are relevant to NES. We present here a description of the non-functional requirements that represent qualities (properties) that are very important to be supported on NES systems. This means that it is important to software infrastructure (operating system, middleware, or else) be able to provide these properties in order to support these systems. The list includes the following properties: dependability, robustness, stability, failure handling, safety, security, privacy, scalability and upgradability.

(a) Dependability, Robustness and System Stability - The notion of dependability includes aspects of reliability and availability. Reliability and availability relate to the

probability of working continuously for a given duration and the percentage of uptime, respectively. Robustness is the ability of the system to perform acceptably when the system operates outside of nominal conditions. Stability is concerned with the system's ability to keep disruptions contained.

(b) Failure handling - In a network embedded system, nodes may fail or experience problems due to several reasons. The failure of individual nodes should not affect the overall task of the network embedded system thus leading to an increased need for providing mechanisms to ensure fault tolerance to the applications.

(c) Safety - This property is concerned with the prevention of the loss of life and/or serious damage to people, property or the environment. This is straightforward for medical devices, but the characteristic is also valid for an aircraft, a car, etc.

(d) Security - This property is concerned with the capability to prevent information and system resources from being used or altered by unauthorized users. A secure system is one where only intended use of the system will be permitted.

(e) Privacy - Privacy is concerned to the ability of an individual or group to isolate themselves or information about themselves and thereby reveal selectively. Privacy is sometimes related to anonymity, the wish to remain unnoticed or unidentified in the public realm.

(f) Scalability - This property indicates the system ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged.

(g) Upgradability - This property is concerned with the degree to which a computer may have its specifications improved by the addition or replacement of components. Most system designers will expect the operating system to make the task easier and to handle some difficult problems like upgrade policy, verification, and security.

Table 1 summarizes the different categories of embedded systems and what properties (requirements) are wished to be found in each of them. In order to establish the importance of the requirements to the various embedded systems they are classified as *mandatory* (M) in case you must fulfill the requirement, *wanted* (W) when it is advantageous to have it, *optional* (O) when have it or not is not a big issue.

Table 1. Properties for different categories of Embedded Systems

EmS/Req.	**(*i*)**	**(*ii*)**	**(*iii*)**	**(*iv*)**	**(*a*)**	**(*b*)**	**(*c*)**	**(*d*)**	**(*e*)**	**(*f*)**	**(*g*)**
SES											
Low-end	M	M	M	M	W	W	M	W	O	O	W
Med-end	W	W	M	M	W	W	W	W	O	O	W
High-end	O	W	M	M	W	W	W	W	W	O	W
NES											
Embedded	M	M	M	M	M	M	M	W	W	W	W
Sensor	M	W	M	M	M	W	W	M	W	M	M
Distributed	W	M	M	M	M	M	M	M	M	M	M

4. Software infrastructure for embedded systems

Embedded systems have been around at least as long as the microprocessor. The software for these systems has been built, more or less successfully, using several different paradigms. Some systems are built from scratch by the manufacturer with all software being created specifically for the device in question. This software may be written in assembly language or may use a higher-level language. Not all components of embedded systems need to be designed from scratch. Instead, there are standard components that can be reused. Software infrastructure, especially standard software components such as operating systems (OS), are examples of these reusable software components. Such components are available from independent software vendors and in some cases as open source software.

The rapid progress in processor and sensor technology combined with the expanding diversity of application fields is placing huge demands on the facilities that an embedded operating system must provide. The variety of applications where embedded systems are being important also implies that the properties, platforms, and techniques on which embedded systems are based can be very different. All the types of embedded systems, from stand-alone embedded systems-SES to Networked embedded systems-NES, need some specific type of service from software infrastructure such as operating system. For instance, these services are supposed to be prepared to attend to functional and non-functional requirements. In this section we give an overview of presently available software infrastructures for embedded systems, particularly regarding operating systems.

Not all embedded systems need to be supported by operating system functionality the simplest embedded systems are usually built without an explicit operating system. Such systems do not have behaviors that require complex mechanisms or real-time scheduling of concurrent tasks, and can therefore be implemented using a simple main loop or executive cyclic approaches. Also, dedicated networked embedded systems can usually be implemented without an operating system, but only with a stand-alone protocol stack.

Embedded systems typically have requirements that are a lot different from the ones for desktop computers, and hence operating systems for embedded systems are diverse from general purpose operating systems (GPOS). Operating systems for embedded systems usually are designed to be tailored for a specific application and therefore are more static than GPOS. Most embedded systems are application specific and require real-time guarantees to function correctly. Therefore, we need operating systems with real-time properties which can be flexibly customized towards the application at hand.

Conventionally, existing operating systems for embedded systems are divided into two categories: embedded operating systems and real-time operating systems [Engel, Heiser, Kuz, Petters and Ruocco 2004]. In this paper we assume that the purpose of the OS can be divided in two categories: real-time embedded systems that can also be called general embedded systems, and domain specific embedded systems which somehow provide specific characteristics for different domains of embedded systems such as automotive, avionics, mission critical systems and wireless sensor networks systems, without loosing OS functionality. For each one of these categories we elaborate a table in order to present how they deal with the embedded systems properties we have presented in section 3. The tables gives you a view of how the systems provide and enforce a certain property: *adequately* (A) in case the system provides and enforces the property, *poorly* (P) when the system provides the property but not as required (for example, when the system is said to be real-time but do not support hard real-time behavior), and *non* (N) when the system does not provide the requirement.

4.1. Embedded and Real-Time Operating Systems

In many embedded systems there is effectively no device that needs to be supported by all versions of the OS, except maybe the system timer. Hence, it makes sense to handle relatively slow devices such as discs and networks by using special tasks (drivers) which are not integrating the kernel of the OS. Protection mechanisms are not always necessary, since embedded systems are typically designed for a single purpose and there is no well defined separation between application and OS functionality. Moreover, many embedded systems are real-time (RT) systems and, hence, the OS used in these systems must be a real-time operating system (RTOS). After all, it is possible to say that some features such as configurability, portability, real-time and reliability, are very desirable in embedded OS

Traditional embedded operating systems like VxWorks [VxWorks 2008], WinCE [MS 2008], QNX [Hildebrand 1992], OS-9, LynxOS, Symbian, are typically large (requiring hundreds of KB or more of memory), general-purpose systems consisting of a binary kernel with a rich set of programming interfaces. Such OSes target systems with greater CPU and memory resources, and generally support features such as full multitasking, memory protection, TCP/IP networking, and POSIX-standard APIs that are undesirable (both in terms of overhead and generality) for sensor network nodes. They provide memory protection given the appropriate hardware support. This becomes increasingly important as the size of the embedded applications grow. In addition to providing fault isolation, memory protection prevents corrupt pointers from causing seemingly unrelated errors in other parts of the program allowing for easier software development. VxWorks, WinCE and QNX are well ranked in the 2005 survey by Embedded System Design [Turley 2005].

Several embedded systems have taken a component-oriented approach for application-specific configurability [Friedrich, Stankovic, Humphrey, Marley and Haskins 2001]. eCos [Massa, 2003] have a goal of lightweight, static composition. These systems consist of a set of components that are wired together (either manually or using a composition tool) to form an application. Components vary in size from fine-grained, specialized objects to larger classes and packages.

Contemporary OS such as Linux, also have extensions that enable them to support real-time applications. These RTOSs such as RTAI [Dozio and Mantegazza 2003] are only suitable for large real-time systems due to footprint required. Other Linux extensions like Embedded Linux [Embedded Linux 2008] and μClinux [μClinux 2008] are more embedded compliant but do not support real-time. These extensions have an architecture which is very much like the Linux architecture, except they do not have to deal with memory management units (MMU).

As embedded systems are getting more and more complex, dynamic, and open, while interacting with a progressively more demanding and heterogeneous environment, the reliability and security of these systems have become major concerns. There is a growing request from stakeholders in embedded systems to make available execution platforms which address both integrity and security concerns. Recently, we have seen some proposals which are really concern with security, reliability, dependability and resilience of operating systems. Basically, they propose to use micro-kernel based operating systems [Liedke 1995] to provide security and dependability for embedded system application. One of the proposals is based on L4 OS [Heiser, Elphinstone, Kuz, Klein and Petters 2007]. Another proposal is based on Minix OS [Tanenbaum, Herder, and Bos 2006].

Table 2 summarizes some operating systems for embedded systems and how they provide the properties (requirements) of section 3.

Table 2. Embedded Systems Requirements provided by OSes for EmS

OS/Req.	(*i*)	(*ii*)	(*iii*)	(*iv*)	(*a*)	(*b*)	(*c*)	(*d*)	(*e*)	(*f*)	(*g*)
VxWorks	P	P	A	A	P	P	P	A	P	A	P
QNX	P	P	A	A	P	P	P	A	P	A	P
Windows CE	P	P	A	P	N	N	N	P	P	P	N
eCos	A	P	A	P	N	N	P	P	P	A	N
pSOS	A	P	P	P	N	N	P	P	N	P	N
RTAI	N	A	P	P	N	N	N	P	N	P	N
uClinux	A	N	A	P	N	N	N	P	N	P	N
L4	P	P	P	A	A	A	P	A	A	A	P
Minix	P	N	P	A	A	A	P	A	A	A	P

Most of the OSes for embedded systems provide the necessary functionalities such as multitasking, memory management, file system, network, etc., through its API. They provide various architecture approaches such as monolithic kernel, micro-kernel based, components based, and library based. However, they do not provide and enforce a lot of non-functional requirements that are necessary in embedded systems, specially the properties usually required in networked embedded systems. Also, most of the OS we listed required large amount of memory to run and do not deal with low power consumption. According to [Turley 2005], the type of operating system that embedded systems developers pick in almost half the cases (44%), is one of the many commercial operating systems or RTOS products for their current project. The remainder was about equally divided among internally developed operating systems, open-source operating systems, and no operating system at all.

4.2. Domain Specific Embedded Operating Systems

This section presents two domains: Avionics and Embedded Sensor Networks which needs specific requirements from software infrastructure such as OS.

Avionics

Significant work has been performed within the avionics domain to achieve the stated objectives. The main body of work has been performed under the banner Integrated Modular Avionics (IMA).

The ARINC 653 [ARINC 1996] is a standard that specifies a programming interface for a RTOS. In addition, it establishes a particular method for partitioning resources over time and memory. ARINC 653 defines an APplication EXecutive (APEX) for space and time partitioning that may be used wherever multiple applications need to share a single processor and memory, in order to guarantee that one application cannot bring down another in the event of application failure. Each partition in an ARINC 653 system represents a separate application and makes use of memory space that is dedicated to it.

The MILS –Multiple Independent Levels of Security and Safety [Alves-Foss, Harrison, Oman and Taylor 2006] approach is proposed to provide a reusable formal framework for high assurance system specification and verification. Separation of kernel is the big issue. In the MILS architecture, the separation kernel only does four very simple things. A MILS kernel is responsible for enforcing data isolation, control of information flow, periods processing and damage limitation policies, and nothing else. Each of these policies counters one or more of the basic foundational threats to system assurance. The MILS separation kernel, has two special characteristics. First, it is the only code that runs in supervisor or privileged mode. No other code, not even device driver code, has the ability to affect the processor's protection mechanisms, particularly the MMU. The second characteristic is that because the separation kernel is so simple it can be very small, approximately 4,000 lines of C language source code.

LynxSecure [DeLong 2007] and PikeOS [Kaiser and Wagner 2007] are examples of MILS compliant OS. They have been used especially in military and avionics industries. Another system, RTEMS [RTEMS 2008] which is not MILS compliant is also used for military and avionics applications. Table 3. summarizes some operating systems for avionics domain embedded systems and how they provide the properties (requirements) of section 3.

Table 3. Embedded Systems Requirements provided by OSes for Avionics Domain

OS/Req.	**(*i*)**	**(*ii*)**	**(*iii*)**	**(*iv*)**	**(*a*)**	**(*b*)**	**(*c*)**	**(*d*)**	**(*e*)**	**(*f*)**	**(*g*)**
LynxSecure	P	P	A	A	A	P	A	A	A	A	A
PikeOS	P	P	A	A	A	P	A	A	A	A	P
RTEMS	P	A	A	A	A	P	A	A	P	P	P

All systems tend to be compliant to ARINC and MILS specifications, where the big issue is Security and Safety. However, some basic requirements for embedded systems such as resource constrains and real-time are not adequately handled. Also, it seems that failure handling is not a very concern for these systems.

<u>Embedded Sensor Networks</u>

Recently, the availability of cheap and small tiny sensors and low power wireless communication allowed the large-scaled deployment of sensor nodes in Embedded Sensor Networks (ESN). An embedded sensor network is a network of embedded computers placed in the physical world that interacts with the environment. These embedded computers, or sensor nodes, are often physically small, relatively inexpensive computers, each with some set of sensors or actuators [Heidemann and Govindan 2004]. Systems of 1,000s or even 10,000 nodes are anticipated. Such systems can revolutionize the way we live and work, it is not irrational to expect that in a decade the world will be covered with wireless sensor networks with access to them via the Internet [Lee, Leung and Son (2008)].

To be usable a sensor networking system must provide several services, further than the lower-level networking primitives. When sensor network programmers make a program for sensor network applications, without any middleware or operating system, the development of the application is so difficult. The sensor networking community typically uses embedded (and, possibly, real-time) versions of existing operating systems such as Linux for the larger devices. These embedded versions provide largely the same programming support as their regular counterparts, but with additional device-level support for embedded controllers, flash memory, and other peripherals specific to these devices. As such, not much research has been required on new operating systems support for these larger devices. On the other hand, the smaller devices (such as the motes) have required novel directions in operating system design.

One such direction has been the development of a number of OSes for embedded sensor networks and networked low-power systems. TinyOS [TinyOS 2008], an operating system for the motes and widely used by many research groups as well as in some segments of industry, deviates significantly from the traditional multi-threaded model of modern operating systems. MantisOS [Bhatti et alli 2005] and Contiki [Dunkels, Grönvall and Voigt 2004] are two recent projects providing multithread support. Other OSes like Nano-RK [Eswaran, Rowe and Rajkumar 2005], and Pixie [Lorincz, Chen, Waterman, Werner-Allen and Welsh 2008] , provide different approaches in order to support embedded sensor networks. Table 4 summarizes some operating systems for sensor networks domain and how they provide the properties (requirements) of section 3.

Table 4. Embedded Systems Requirements provided by OSes for Sensor Networks

OS/Req.	**(*i*)**	**(*ii*)**	**(*iii*)**	**(*iv*)**	**(*a*)**	**(*b*)**	**(*c*)**	**(*d*)**	**(*e*)**	**(*f*)**	**(*g*)**
TyniOS	A	P	A	P	P	N	N	N	A	A	N
Contiki	A	P	A	P	P	N	N	N	A	A	P
MantisOS	A	P	A	P	P	N	N	N	P	A	A
Nano-RK	A	A	A	P	P	N	N	N	P	P	P
Pixie	A	P	A	P	P	N	N	N	P	P	A

One of the characteristics of OSes for sensor networks is to be adequately able to deal with resource constraints such as memory and low power. They all provide and ensure this particular requirement. However, most of the systems does not provide an adequately real-time behavior. Regarding some of the important requirements for distributed embedded systems these OSes perform very poorly. Therefore, it seems that those requirements have to be supported in a different level than operating system.

Conclusion

We have analyzed operating systems that are supposed to be tailored for different types of embedded systems, from stand alone embedded systems to distributed real-time embedded systems. The analysis was carried out based on documentation available about the systems, and the results are shown as an informal rank which consider how the systems are capable of supporting the required properties.

The analysis have shown that most of the OSes for embedded systems do not provide or support various non-functional requirements, specially the properties usually required in networked embedded systems. In the case of avionics most of the OSes are specially concern about Security and Safety. However, they do not attend adequately some basic requirements such as real-time and failure handling. For sensor networks systems, the analysis shown that most OSes are very concern with supporting resource constrains requirement specially memory and low power. Unfortunately, most of them do not provide adequately real-time behavior. In addition, regarding some of the important requirements for distributed embedded systems they perform very poorly.

The analysis also shown that, specially regarding NES, OSes do not provide most of the (non-functional) properties they required. In order to support the required properties (requirements) the systems have been built using another layer of software infrastructure, the middleware. We believe that one of the challenges of OS infrastructure for NES is to provide support for these required properties.

References

Alves-Foss, J., Harrison W.S., Oman, P. and Taylor, C. (2006), The MILS Architecture for High-Assurance Embedded Systems. International Journal of Embedded Systems, 2 , pages 239-247.

ARINC (1996), *ARINC 653:* Avionics Application Software Standard Interface (Draft 15). Airlines Electronic Engineering Committee (AEEC), June 17th, 1996.

Bhatti, S. et alli (2005), MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms, ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks, vol. 10, no. 4, August, pages 563-579.

Crespo, A., Ripoll, I., González-Harbour, M. and Lipari, G. (2008). Operating System Support for Embedded Real-Time Applications, EURASIP Journal on Embedded Systems, Volume 2008 , 2 pages.

DARPA IXO (2002), Networked Embedded Software Technology (NEST). (http://www.darpa.mil/ixo/) .

DeLong, R.J. (2007), LynxSecure Separation Kernel – a High-Assurance Security RTOS, LynuxWorks, San Jose, CA. http://www.lynuxworks.com.

Dozio, L. and Mantegazza, P. (2003),"Real Time Distributed Control Systems Using RTAI", In: Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Hakodate, Hokkaido, Japan, 14-16 May.

Dunkels, A., Grönvall, B. and Voigt, T. (2004),"Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors", In: Proceedings of the First IEEE Workshop on Embedded Networked Sensors, Tampa, Florida, USA, November.

Embedded Linux 2008, http://www.linuxdevices.com/.

Engel, F., Heiser, G., Kuz, I., Petters, S. and Ruocco, S. (2004), "Operating systems on SoCs: a good idea?", In: Embedded Real-Time Systems Implementation (ERTSI 2004) Workshop, Lisbon, Portugal, December.

Eswaran, A., Rowe, A. and Rajkumar, R. (2005),"Nano-RK: An Energy-Aware Resource-Centric Operating System for Sensor Networks", IEEE Real-Time Systems Symposium, December.

FP6-IP-RUNES (2005) - D5.1 Survey of Middleware for Networked Embedded Systems, January 2005 – (http://www.ist-runes.org/docs/deliverables/D5_01.pdf).

Friedrich, L., Stankovic, J., Humphrey, M., Marley, M. and Haskins, J.(2001), A survey of configurable component-based operating systems for embedded applications, IEEE Micro, May, pages 54-68.

GENI (2006), Report of NSF Workshop on Distributed Real-time and Embedded Systems Research in the Context of GENI (October 2005 and February 2006) – (http://www.geni.net/GDD/GDD-06-32.pdf).

Heidemann, J. and Govindan, R. (2004), An Overview of Embedded Sensor Networks, In: Handbook of Networked and Embedded Control Systems, Springer-Verlag.

Heiser, G., Elphinstone, K., Kuz, I., Klein, G. and Petters, S. (2007), Towards trustworthy computing systems: taking microkernels to the next level, ACM SIGOPS Operating System Review, 41(4), July, pages 3-11.

Hildebrand, D. (1992), "An Architectural Overview of QNX", In: Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures, pages 113–126.

Kaiser, R. and Wagner, S. (2007), The PikeOS Concept History and Design, SYSGO, http://www.**sysgo**.com.

Lee, I., Leung, J. and Son, S. (2008). Handbook of real-time and embedded systems, Chapman & Hall/CRC computer & information science series, 2008.

Liedke, J. (1995), "On μ-Kernel Construction", In: Proceedings of 15th ACM Symposium on Operating Systems Principles, December, pages 237-250.

Lorincz, K., Chen, B., Waterman, J., Werner-Allen, G. and Welsh, M. (2008), "Pixie: An Operating System for Resource-Aware Programming of Embedded Sensors", Fifth Workshop on Embedded Networked Sensors (HotEmNets'08), June.

Massa, A. (2003), Embedded Software Development with eCos, Prentice Hall.

MS (2008), Microsoft Windows CE, http://www.microsoft.com/windowsce/embedded/

NRC (2001), Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers. Committee on Networked Systems of Embedded Computers, National Research Council, 2001

RTEMS (2008), http://www.rtems.com/.

Tanenbaum, A., Herder, J. and Bos, H. (2006), Can we Make Operating Systems Reliable and Secure?, IEEE Computer, May, pages 44-51.

TinyOS 2008, http://www.tinyos.net/ .

Turley, J. (2005), Embedded systems survey: Operating systems up for grabs, Embedded Systems Design, http://www.embedded.com/columns/surveys.

VxWorks (2008), http://www.windriver.com.

Yodaiken, V. and Barabanov, M. A Real-Time Linux. http://www.soe.ucsc.edu/~sbrandt/courses/Winter00/290S/rtlinux.pdf

μClinux 2008, Embedded Linux Microcontroller Project – http://www.uclinux.org/.