Real-time Operating Systems for Self-coordinating Embedded Systems

Franz J. Rammig, Marcelo Götz, Tales Heimfarth, Peter Janacik, Simon Oberthür
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
D-33102 Paderborn
{franz, mgoetz, tales, pjanacik, oberthuer}@uni-paderborn.de

Abstract: It can be observed that most technological artefacts are becoming intelligent "things that think" and most of these intelligent objects will be linked together to an "Internet of things". To master this omnipresent virtual "organism", completely new design and operation paradigms have to evolve. In this paper we discuss how research of our group at the University of Paderborn is providing fundamental principles, methods, and tools to design real-time operating systems for this virtual "organism" of the future. Based on our fine-granular library for the construction of reflexive RTOS, the necessary configuration tool and its on-line version are discussed. Next step towards self-coordination is a profile management system to support self-optimization of the RTOS. The included flexible resource manager allows migration of RTOS services dynamically between programmable processors and re-configurable HW. In a final step the RTOS itself can be distributed. Its services are provided by a cluster of instances instead of a single one. This makes a sophisticated dynamically self-optimizing communication system necessary.

1 Introduction

In the world of information technology it is no longer the computer in the classical sense where the majority of IT applications are executed; computing is everywhere. More than 20 billion processors have already been fabricated and the majority of them can be assumed to still be operational. At the same time virtually every PC worldwide is connected via the Internet. This combination of traditional and embedded computing creates an artefact of a complexity, heterogeneity, and volatility unmanageable by classical means. Metaphorically speaking, it may be treated as an organism made up of computers, networks, system software, applications, and, most importantly, human beings. This virtual organism as it exists today is still in a state of adolescence. Each of our technical artefacts with a built-in processor can be seen as a "Thing that Thinks", a term introduced by MIT's Thinglab. It can be expected that in the near future these billions of Things that Think will become an "Internet of Things", a term originating from ETH Zurich. This means that (using the above metaphor) we will be constantly surrounded by a virtual "organism" of Things that Think. This organism cannot be managed by any traditional means. As a consequence, completely new RTOS approaches are needed. The virtual organism is volatile by

nature. Its constituents are mobile and adapt to changing environmental contexts in a self-organizing manner. This means that the RTOS itself has to be self-adapting as well, which includes that it has to be reflexive. Techniques are needed to allow such an RTOS to sense its environment, reflect this information at its present state, decide about self-modifications and execute this self-modification. Some approaches developed in our group to deal with these challenges will be sketched out in this contribution. These approaches are based on the assumption that system services are provided by means of alternative profiles. An intelligent profile manager selects the presently best suited collection of services and their respective profiles. Re-configurable hardware is commercially available and introduces an additional degree of potentials and at the same time of volatility. Systems consisting of programmable microcontrollers and re-configurable HW can be tailored dynamically towards actual demands by application software. It depends on the actual characteristics of application load whether processor capacity or FPGA area are the more valuable good. This means that RTOS services should migrate dynamically between SW and HW implementation. An additional challenge arises if the global interaction of all these objects is envisioned. Together with potential mobility of the devices, a degree of volatility is introduced that is unknown in traditional systems. Centralized approaches seem to be completely inadequate to cope with this challenge. We therefore developed a cross-layer communication system that uses decentralized techniques to provide a communication backbone in highly volatile environments. These techniques are inspired by the behaviour of ant colonies. If such a self-adapting and self-optimizing communication backbone has been established, a new perspective on the RTOS becomes possible. There may be no longer a need to provide all potentially requested services by each instance of the RTOS on the various nodes. In principle it would be sufficient to provide the services by a complete cluster of instances as a whole. However careful tailoring of service distribution is necessary as any remote service provisioning is very costly due to the extremely high power consumption of communication.

2 Related Work

Some work on customization and configuration as approach for adapting a system to the requirements of an application has been reported in literature. In the field of (real-time) operating systems (OS) various approaches for customization have been proposed. SYNTHESIS [Mas92] adapts its code by partially evaluating and recompiling condition statements depending on available input data at run time. This can result in changing compare instructions and conditional jumps by unconditional jumps and vice versa. Likewise, the OS K42 [SAH+03] includes system support for online reconfiguration by a hot-swap mechanism using C++ virtual function tables.

Reconfigurable hardware/software based (hybrid) architectures are being very attractive for implementation of run-time reconfigurable embedded systems. Moreover, the hardware-software allocation of application tasks to dynamically reconfigurable embedded systems (by means of task migration) allows for customization of their resources during run-time to meet the demands of executing applications, as can be seen in [HMM04] and [MNC+03].

Nevertheless, the current research efforts spent in this field are just focused on application level, leaving to the RTOS the responsibility to provide the necessary mechanisms and run-time support (see [BJS04], [WK01] and [NCV+03]). In [BJS04], e.g., the resources are dynamically arranged to support dynamic application tasks arriving into the system. However, the RTOS itself is still static.

In our novel proposal, we expand and adapt those concepts to the RTOS level. Thus, not just the application but also the RTOS itself may also be reconfigured over a hybrid architecture.

Current OS are not able to work on top of ad hoc networks in a distributed manner (e. g. *VxWorks* [Sys99], *Apertos* [Yok92]). OS for sensor networks like *TinyOS* [ea00] do not support complex, distributed applications using elaborate OS services like our approach.

Several of our approaches are based on the *division of labour in ants* [BDT99]. Conducted experiments suggest that resilience of a colony as a whole is determined by the *elasticity* observed at the individual level, which is modelled as follows: *Intensity of stimulus* s associated with a particular task, determined by quantitative cues sensed; *response threshold* θ, determining the tendency of an individual to respond to a stimulus. Finally, n describing the *steepness* of the threshold. They are combined in the *response function*: $T_\theta(s) = \frac{s^n}{s^n+\theta^n}$.

3 Basic Principles of the DREAMs Library and the TereCs Configuration System

OS and run-time platforms even for heterogeneous processor architectures can be constructed from customizable components (*skeletons*) out of the DREAMS's (**D**istributed **R**eal-time **E**xtensible **A**pplication **M**anagement **S**ystem) library [Dit99]. By creating a configuration description all desired objects of the system have to be interconnected and afterwards fine-grained customized. The goal of that process is to add only those components and properties that are required by the application. The creation of a final configuration description for DREAMS was automated during the project TEReCS (**T**ools for **E**mbedded **Re**al-Time **C**ommunication **S**ystems) [Bök03]. During that project a methodology was developed in order to synthesize and configure the OS for distributed embedded applications.

TEReCS distinguishes between knowledge about the application and expert knowledge about the customizable OS. Knowledge about the application is considered as a requirement specification. This specification is input to the configurator and abstractly describes the behavior of the application and some constraints (deadlines), which have to be assured. The behavior of the application is defined by the OS calls it requests.

The complete and valid design space of the customizable OS is modelled by an AND/OR service dependency graph in a knowledge base. This domain knowledge contains options, costs, and constraints and defines an over-specification by containing alternative options. The configuration process removes some domain specific knowledge by exploiting knowl-

edge about the application. Thereby, a configuration for the run-time platform will be generated. In the AND/OR graph nodes represent *services* of the OS and are the smallest atomic items, which are subject to the configuration. Mandatory dependencies between services are specified by AND edges, optional or alternative are specified by OR edges. Services and their dependencies have costs and can be prioritized. *Constraints* for the alternatives can be specified. Root nodes of the graph are interpreted as *system primitives*/*system calls* of the OS. Each of these primitives points to one concrete service. The service dependencies span a complete graph. The leaf nodes can refer to hardware devices. These devices are communication devices, which again refer to communication media.

4 Self-Optimization

For building an RTOS system, which optimizes itself and provides the released resources optimally to the applications, we have to answer the three following questions: First, how can we get information about the dynamic resource requirements during run-time of an application? A suitable interface between the application and the OS is required. Second, how can we describe the design space for reconfiguration of the RTOS system? A model is required that describes the dependencies between the system services. Third, based on this information: Which is an adequate system configuration? A resource management system is required to activate/deactivate the system services and to release resources for the applications.

Our self-optimizing RTOS consists of three components Profile Framework, Online TEReCS and Flexible Resource Management (FRM), which are briefly described in this section. A concrete example and a prototype are provided in [OBG05].

Profile Management

The Profile Framework, described in detail in [OB04], has two main purposes: The first purpose is to model information about the dynamic resource requirements of the application and of the system services. The knowledge about the resource requirements of the applications is necessary to optimize the system by deactivating services that are currently not required. The resource requirements of the system services are necessary to determine the amount of resources, which can be released and to guarantee that applications and system services get their required resources under hard real-time conditions. The second purpose is to provide released resources from deactivated system services to the applications. By means of this framework the developer can define a set of profiles per application. Profiles describe different *service levels* of the application, including different quality and different resource requirements. Additionally each profile holds information about the time to activate and deactivate the profile. All resource allocations of an application require an announcement to the FRM. The *maximal assignment delay*, specified in each profile per resource, is the worst case time the assignment of the resource can be delayed from the announcement until its allocation.

Online Configuration System: Online TEReCS

The Online TEReCS component has knowledge of the design space of the customizable components, models system services as profiles, and is responsible for the low-level reconfiguration of the system services. For the online case of the configuration a coarse grained hierarchy of this graph is used on the system service level – one service can be composed of many components. Thus, the overall decision space is more restricted and the number of generated profiles is easy to manage.

For a system service, which can be in a deactivated or in an activated state, two profiles are created. One profile for the activated and one profile for the deactivated state. In the profile for the activated state the service specifies the required resources to fulfill its task. In the profile for the deactivated state the service occupies the resources it provides in the activated state. Hereby, it is guaranteed that the services can only be deactivated if the provided services are not required.

Flexible Resource Manager

The major goal of the FRM [OB04] is to optimize the resource utilization and the over-all system quality by selecting profiles for activation under the actual conditions. To maximize the utilization the FRM puts the resources, which are held back for worst case resource requirements of an application, at other application's disposal. Normally, an application acquires as many resources as it requires for worst-case scenarios. Thus, the application has always enough resources for its tasks and can fulfill its service at any time. These resources are only required when the worst-case scenario occurs. In our approach the FRM tries to minimize this internal waste of resources, by making these resources temporarily available to other applications under hard real-time conditions. The FRM is responsible for switching between the profiles of the applications and system services under the defined switching conditions. The decision is made based on a quality function, which considers the actual resource requirements, the importance of an application, and the quality of the profiles. To guarantee hard real-time conditions, an acceptance test for profile activation is integrated in the optimization process.

5 Reconfigurable Hardware/Software-based RTOS

Differently from the normal approach where the design of such RTOS is done offline, the proposed approach suggests the use of new reconfigurable architectures (e.g. Virtex-II Pro^{TM}FPGA) in order to support the development of a hardware/software reconfigurable OS [Göt04]. In this proposed architecture, the Real-Time OS (RTOS) is capable to adapt itself to current application requirements, tailoring the RTOS components for this purpose. Therefore, the system continuously analyzes the requirements and reconfigures the RTOS components at the hybrid architecture optimizing the use of system resources. Hence, the system is capable to decide on-the-fly which RTOS components are needed and also to which execution environment (CPU or FPGA) they should be assigned. Our system concept is based on a microkernel approach, where the RTOS services are provided as a set of hardware and software components. These components can, during run-time, be reallocated (reconfigured) over the hybrid architecture. The usage of microkernel also in-

corporates the nature advantage of flexibility and extensibility (among others) of it, which is very desired in our case in order to better perform the reconfigurability aspects.

RTOS Service Assignment

The decision of which RTOS service is to be placed in hardware or software, is based on the amount of resources and the communication costs needed by each service. If an RTOS component is allocated at the software environment, it will usually increase the processor load. Otherwise, at the hardware environment it will require some amount of circuit area available. Similarly, the communication requirements of each service will infer in different communication costs depending on its placement. To solve this assignment problem, a cost function has been proposed [GRP05], which tries to minimize the total amount of resource used by the RTOS component set currently needed by the application. Besides the limited amount of resources available for the RTOS, a weighted load balancing of the resources used is specified as a system constraint. Thus, it is possible to influence the placement decision based on the current valuable goodness of the resources required, since the resources are shared between RTOS and application software. Moreover, through a correct load balancing, it can be avoided that one execution environment gets near to its full utilization and so, making the system capable to absorb variations of the application demands.

System Reconfiguration

As the application requirements are considered to be dynamic, the component costs are not static and depend on the current application demands. This leads to the fact that a certain system configuration (component allocation) may no longer be valid after application changes. Therefore, a continuous evaluation of the components partitioning is necessary. Whenever the system reaches an unbalanced situation, the RTOS components should be reallocated in order to bring the system again into the desired configuration. In this situation, not just the new assignment problem needs to be solved again, but also the costs (time) necessary to reconfigure the system from the current state to new one need to be evaluated. This evaluation is necessary since we are dealing with real-time systems and so, it has a limited time available for reconfiguration activities (which depends on the current application time constraints). For this case, our approach consists on scheduling the RTOS components reconfiguration together with the application tasks. Therefore, techniques based on RTOS scheduling theory, like fixed/dynamic priority servers, are used to provide a safely time for reconfiguration activities and still guarantee the correct application tasks timeliness.

6 Self-optimizing Communication System

In the introduced virtual “organism” made up of mobile constituents, wireless communication plays an important role. It is the logical next step after having considered wired communication in our previous work, which is only suited for static networks. Given its relatively high cost, the means of wireless communication has to be utilized economically.

Employed within this context, self-organization promises lower complexity and communication overhead, as well as, increased robustness and good scalability properties. We aim to use this powerful paradigm at all levels of the network stack. Our efforts are predominantly directed towards the following aspects: topology control, combined link metrics, as well as, adaptive ant colony-based multipath routing.

Network regions with higher node density tend to lead to a greater amount of energy waste through overhearing, collisions, and retransmissions after frame losses, while not increasing the network's capacity. Therefore, we develop an emergent *topology control* reducing the number of active nodes in such areas. It divides nodes in *transporters*, actively participating in network operation (e. g. part of a routing backbone), and *workers*, not involved in these activities. The underlying mechanism for state transitions is motivated by the *division of labour in ants*. State changes are realised using a response function, as reviewed in Section 2. The underlying idea behind the response function for the *worker-to-transporter transition* is to use a threshold function that decreases the tendency for a worker to become a transporter with more energy consumed and a lower density of neighboring worker nodes. The stimulus rises with a longer disconnection time from the backbone. On the other hand, the underlying idea of the response function for the *transporter-to-worker transition* is to use a threshold that decreases, when the backbone around a transporter becomes denser. The stimulus rises with the amount of energy consumed since becoming transporter.

Efficient utilisation of the wireless medium can only be realised given a good estimate of the "real" quality (in terms of the expected reception success) of the underlying links. Based thereon, protocols and applications at upper layers are given the means to adjust their optimization strategies. None of the basic, conventional metrics provides fully satisfying results: Received signal strength measurements are rough, but quick estimates. The observation of transmission successes yields precise results, but only after enough data is provided. Neither of these metrics recognises volatile links, so that upper layers could avoid them. Hence, we use a *combined link metric*, the so called *virtual distance*, incorporating received signal strength, reception success rate, and a history component.

At the network layer, we extend state-of-the-art *ant colony-based multipath routing* by several elements. Our choice for ant colony-based routing was motivated by its promising characteristics: utilisation of multiple paths, probabilistic decisions, consideration of lower-layer feedback, and local work, helping to achieve more resilience, a higher degree of load balancing, and a lower communication overhead. To accelerate the transitory phase in ant-colony routing, we propose a novel *waycost function* enabling pheromone adjustments that more accurately reflect underlying properties. To enable this, packets carry a field containing the combined link metric-based, accumulated waycost taking into account all visited (i.e. utilized) links. The amount of pheromone deposited on a packet's way then decreases exponentially with increasing waycost. In order to reduce the frequency of expensive route failures and reconstructions due to changes incurred by topology control, we further propose a fuzzy distinction between transporters and workers: during a next-hop decision, the probability for choosing a transporter is increased.

In ns-2 [The] simulations, we demonstrated the scalability in terms of node number and density, as well as, the stability of the results of topology control. Further, the results

showed a significant improvement of the quality of utilised links using the combined link metric.

7 NanoOS – Service Provisioning by Clusters of OS Instances

In an "Internet of things" scenario, heterogeneous sorts of devices are connected through different network infrastructures. Vantages can be achieved with this scenario with the introduction of distributed computing instead of bare centralized one. We can generalize the idea of self-optimization from one node to a network of connected devices. In our approach, we are considering small devices connected through an ad hoc network. In order to support complex functionality in constrained nodes, the OS uses a novel approach: it distributes the services among the nodes. This means that each node of the system has just a small part of the kernel of the complete OS; a group of nodes together form an instance of the OS. Thereby the resource requirements in each node are much less than on a node with all services locally instanced. The prototype of an OS, built over the self-optimizing communication system presented in the previous section, enables this vision. It is called NanoOS [HR04].

In the NanoOS, each application has one goal and is composed by a set of tasks; tasks are the atomic unit of the distributed application. The developer is responsible for the division of the application in several cooperative tasks. For this the OS supports hierarchic grouping of tasks. The services and tasks can migrate in order to optimize the communication. Each service is autonomous, it is an agent that tries to find a good position in the network, a node with enough resources and where the communication with the clients (application tasks) is minimized.

Resource-aware Clustering

The distribution of the services brings several problems. The consistency of the kernel in each node should assure also a global consistency of the entire "distributed" kernel (global consistency). The connection quality plays an important role in this case. In order to restrict the traffic between nodes with good connection quality and also to reduce significantly the organization overhead of the OS (small routing table, service discovery scope and consistency control), the entire system is divided in clusters. Each cluster should contain the necessary resource for all services running inside the cluster. This brings the concept of a complete OS instance per cluster. ***Cluster***: set of nodes where all the services required by the tasks in this set are available in nodes inside the set.

We developed a heuristic that uses just a small amount of local communication to find the clusters that contain a minimum amount of resources. As result of our clustering algorithm, each node will be assigned to a cluster $\phi \in 0, ..., n-1$ where n is the number of clusters. During the clustering, a node can be of type *clusterhead* (task that represent a cluster), *member* (ordinal member of a cluster) or *not clustered* (task that does not belong to any cluster). The main idea of the algorithm is to use the principle of the stimulus/threshold found in the division of labor of some social insects: members of a nest with different

morphology have different threshold to do a determined task. In our case, nodes with good communication and plenty neighborhood have a smaller threshold to become *clusterhead* than nodes with few loosely connected neighbors. The stimulus to become clusterhead increases with the time that a node is of *not clustered* type.

Service Distribution

The distribution algorithm is responsible for deciding where each service of the OS should be placed inside a cluster. The algorithm is executed in each service that is independent to decide the destination node. The main goal of the distribution algorithm is to reduce the global communication overhead (in a continuous self-optimization process), therefore the state of the wireless links is an important parameter used in the optimization process. For this reason, we used the presented link metric that rates the connection quality of the ad-hoc links (*virtual distance*). The positioning of the services is done based on this virtual distance graph and the resource availability in each node. It tries to minimize the sum of all virtual distances of all remote system calls. This is an *NP-Complete* combinatorial problem with similarities with the QAP (*Quadratic Assignment Problem*). To solve it distributively, we developed an heuristic based on the principles of the *swarm intelligence*, more specifically, *Ant Colony Optimization* [BDT99]. Each service of the OS is modeled as a food source and the calls coming from different nodes are the ants looking for the service. Each node has a pheromone table. In each node where an ant passes when going in the direction of the food, it increases the pheromone of correspondent service in the same proportion to the communication cost of the request. The optimization is done when the service (food source) **moves** in the direction of the higher pheromone; that means the service migrates in the direction from where more calls are coming.

8 Conclusion

With the library for real-time operating services called *DREAMS* we are providing a sound basis for the construction of tomorrow's real-time operating systems. We extended this fundamental component-based library by means of a configuration system that can be applied even in on-line mode. Strategic decisions are made by a profile management system, supported by a flexible resource manager. Our reflexive RTOS is able to migrate services dynamically between SW and HW implementations to make sure that always the actually most valuable resources are available for application processes. A fully decentralized, self-organizing communication system for ad-hoc networks, based on ant colony inspired techniques opens the path to highly distributed, highly dynamic systems. This communications system allows even to distribute RTOS services over a cluster of instances of the RTOS in such a way that the total functionality of the RTOS is no longer provided by each individual instance but by the cluster in total.

In the future this innovative approach will be extended further. More sophisticated learning algorithms and even more biologically inspired techniques will be included. The solution will be applied in a variety of applications ranging from sensor networks via cooperative mini-robots up to swarm-based space missions.

References

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, NY, 1999.

[BJS04] Krishnamoorthy Baskaran, Wu Jigang, and Thamipillai Srikanthan. Hardware Partitioning Algorithm for Reconfigurable Operating System in Embedded Systems. In *Sixth Real-Time Linux Workshop*, 2004.

[Bök03] C. Böke. *Automatic Configuration of Real-Time Operating Systems and Real-Time Communication Systems for Distributed Embedded Applications*. Phd thesis, Paderborn University, Paderborn, Germany, 2003.

[Dit99] C. Ditze. *Towards Operating System Synthesis*. Phd thesis, Paderborn University, Paderborn, Germany, 1999.

[ea00] J. Hill et al. System architecture directions for networked sensors. In *Proc. of the 9th int. conf. on architectural support for programming languages and OS*, 2000.

[Göt04] Marcelo Götz. Dynamic Hardware-Software Codesign of a Reconfigurable Real-Time Operating System. In *ReConFig*. Mexican Society of Computer Science, 2004.

[GRP05] Marcelo Götz, Achim Rettberg, and Carlos E. Pereira. Towards Run-time Partitioning of a Real Time Operating System for Reconfigurable Systems on Chip. In *Proc. of IESS*, Manaus, Brazil, 2005.

[HMM04] J. Harkin, T. M. McGinnity, and L. P. Maguire. Modeling and optimizing run-time reconfiguration using evolutionary computation. *Trans. on Embedded Computing Sys.*, 2004.

[HR04] T. Heimfarth and A. Rettberg. NanoOS - Reconfigurable OS for Embedded Mobile Devices. In *In Proc. of the International Workshop on Dependable Embedded Systems*, Florianopolis, Brazil, 2004.

[Mas92] H. Massalin. *Synthesis: An Efficient Implementation of Fundamental Operating System Services*. Phd thesis, Columbia University, 1992.

[MNC+03] Jean-Yves Mignolet, Vincent Nollet, Paul Coene, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. Infrastructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-Chip. In *DATE*, 2003.

[NCV+03] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Designing an Operating System for a Heterogeneous Reconfigurable SoC. In *Proc. of IPDPS*, 2003.

[OB04] S. Oberthür and C. Böke. Flexible Resource Management - A framework for self-optimizing real-time systems. In B. Kleinjohann, Guang R. Gao, H. Kopetz, L. Kleinjohann, and A. Rettberg, editors, *Proc. of IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES'04)*. Kluwer Academic Publishers, 23 - 26 August 2004.

[OBG05] S. Oberthür, C. Böke, and B. Griese. Dynamic Online Reconfiguration for Customizable and Self-Optimizing Operating Systems. In *Proc. of the 5th ACM international conference on Embedded software (EMSOFT'2005)*, pages 335–338, 18 - 22 September 2005. Jersey City, New Jersey.

[SAH+03] C. Soules, J. Appavoo, K. Hui, D. Silva, G. Ganger, O. Krieger, M. Stumm, R. Wisniewski, M. Auslander, M. Ostrowski, B. Rosenburg, and J. Xenidis. System Support for Online Reconfiguration, 2003.

[Sys99] Wind River Systems. VxWorks 5.4 - Product Overview, June 1999.

[The] The network simulator. Online. http://www.isi.edu/nsnam/ns/, accessed April 8, 2005.

[WK01] Grant Wigley and David Kearney. The Development of an Operating System for Reconfigurable Computing. In *FCCM*, 2001.

[Yok92] Y. Yokote. The Apertos Reflextive Operating System: The Concept and Its Implementation. In *OOPSLA Proceedings*, 1992.