Dynamic Online Reconfiguration for Customizable and Self-Optimizing Operating Systems [*]

Simon Oberthür, Carsten Böke, Björn Griese
Heinz Nixdorf Institute
University of Paderborn
Fürstenallee 11
D-33102 Paderborn, Germany
simon@oberthuer.net, cboeke@upb.de, bgriese@hni.upb.de

ABSTRACT

When applications adapt their behavior to the requirements of the environment, their resource usage can change dramatically. The resource usage implies the services that the applications require from the operating system. Thus, the operating system must either provide all services that are totally required over time or reconfigure itself. Reconfiguration of the operating system means to support on demand services or the possibility to degrade services. We present an approach where we extend our offline customizable operating system in order to be dynamically reconfigurable during run-time. Additionally, we describe the procedure how the operating system is aware of the current required services. We claim that the resource usage between the applications and the operating system is optimized. Thus, we derive a **self-optimizing real-time operating system** (SO-RTOS). This work concentrates on the *integration* of the *configurator*, which models the design space and controles the low-level reconfiguration, and the *resource manager*, which is responsible for the timeliness and optimality. An optimization case study realized on a prototype validates our approach.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Real-time systems and embedded systems

General Terms

Design, Management, Performance

Keywords

Real.time operating system, self-optimizing

1. INTRODUCTION

In the area of distributed systems a new trend can be observed. Applications adapt themselves to the requests from their environment. Therefore, applications support reconfiguration of their services. This reconfiguration helps to improve the application's QoS, which can lead into an improvement of the system's overall quality. Besides this aspect, the resource usage can be optimized. Thereby, resources that are only required for high level QoS functions can be released, when a lower level QoS is satisfying the currently requested constraints.

When the applications change their behavior and their resource requests dynamically during run-time, then the underlying operating system (OS) should reconfigure also its QoS in form of the current provided services. For example, a specific protocol stack should only be present in the OS, when applications request this protocol for their communication. I.e., that a reconfigurable/customizable OS includes only those services that are currently required by its applications. Hence, services of the OS must be loaded or removed on demand. Thus, the OS also releases valuable resources that can be used by other applications.

In embedded systems, the reconfiguration is critical. These systems often run under hard or soft real-time constraints. The real-time operating system (RTOS) always has to assure a timely and functional correct behavior and has to support the required services. Thus, the reconfiguration underlies the same deadlines as the normal operation of the applications. To handle exactly this problem, we reuse our real-time capable *Profile Framework* and *Flexible Resource Manager*. It models the reconfiguration and the transaction of the reconfiguration under real-time constrains. An acceptance test for the reconfiguration requests assures these constraints.

2. PREVIOUS WORK

2.1 TEReCS

Operating systems and run-time platforms for even 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 [4]. By creating a configuration description all desired objects of the system have to be interconnected and afterwards fine-grained customized. The primary goal of that process is to add only those components and properties that are really 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) [1]. During that project a methodology was developed in order to synthesize and configure the operating system for distributed embedded applications.

TEReCS distinguishes strictly between knowledge about the application and expert knowledge about the customizable operating

[*] This work was developed in the course of the Special Research Initiative 614 "Self-optimizing Concepts and Structures in Mechanical Engineering" – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

system. Knowledge about the application is considered as a requirement specification. This requirement specification is input to the configurator. The requirement specification 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 operating system calls it requests. It specifies which process calls which primitive at what time. Especially the communication channels between the processes have to be specified including their properties (max. data size, period, etc.).

The complete and valid design space of the customizable operating system is specified by a so-called AND/OR service dependency graph in a knowledge base [3]. 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 knowledge about the application. Thereby, a configuration for the run-time platform will be generated. The integration of the domain and application knowledge defines a knowledge transfer from the application down to the operating system.

The complete valid design space of the configurable operating system is specified by an AND/OR graph:

- Nodes represent *services* of the operating system and are the smallest atomic items, which are subject to the configuration
- Mandatory dependencies between services are specified by AND edges
- Optional or alternative dependencies between services are specified by OR edges
- Services and their dependencies have costs and can be prioritized
- *Constraints* (preferences, prohibitions, enforcements under specific conditions) for the alternatives can be specified
- Root nodes of the graph are interpreted as *system primitives*/*system calls* of the operating system

The main objective of the configuration process is to remove all OR dependencies from the graph (over specification $\rightarrow$ complete and non-ambiguous specification). The configuration can be interpreted as a sub-graph without any alternatives.

The *system primitives* are the root nodes of the service dependency graph. Each of these primitives point 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.

The algorithm works, e. g. for communication primitives, as follows: A path can be found through the complete graph from the sending primitive down to the sending device, considering the routing and then up to the receiving primitive. The services that are visited on this path have to be installed on the appropriate nodes of the service platform. Thereby, the path should create minimal costs by the use of the services.

Such paths will be searched for all primitives that are used in the requirement specification. Because only a subset of all primitives is normally used, especially the particular selection is responsible for the instantiated services and its parameterization. The primitives can be considered as the strings of a puppet. Depending on which strings are pulled, the "configuration" of the puppet will change accordingly. The service dependencies can be compared to the joints of the puppet. The algorithm is named *"Puppet Configuration"* [1].

2.2 Flexible Resource Manager

Part of our *Flexible Resource Manager* (FRM) is the *Profile Framework*. We only give a brief overview about the concepts within this section. A formal description can be found in [7].

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. The resource manager then tries to find an appropriate *resource assignment* at run-time which optimizes the system behavior and resource utilization.

A single profile contains the following information:

Resource requirements: The *maximum* and *minimum resource usage* per resource of the application when the profile is active.

Maximal assignment delay: All resource allocation of an application require an announcement to the FRM. The maximal delay is the worst case time the assignment of the resource can be delayed by the FRM from the announcement until its allocation.

Switching conditions: The information between which profiles can be switched and the worst-case execution times (WCET) of the enter and leave functions of the profiles.

Profile quality: By help of this value the profiles of an application can be ordered according to their quality. So the FRM knows which profile to prefer when trying to increase the system quality by selecting a profile for activation.

The major goal of the FRM is to optimize the resource utilization and the over-all system quality by selecting profiles for activation under the current conditions. To maximize the utilization the FRM puts the resources that are held back for worst case resource requirements of an application at other application's disposal. Normally, an application acquires as much resources as it requires for worst-case scenarios. On this base the application has always enough resources for its tasks and can fullfill its service at any time. Yet, these resources are only required when the worst-case scenario occurs. Thus, the FRM also tries to minimize this internal waste of resources.

Actual, the FRM framework extends the ideas of the ARM middleware [5]. While the latter assumes a system-wide defined constant switching time, FRM supports transition specific WCET times and additionally supports temporary *over-allocation* of the resources. The definition and design of the profiles for the FRM with their resource usage is already integrated into the design phase for embedded applications [2].

3. DYNAMIC RECONFIGURATION

The main idea is to release resources of system services by deactivating or activating basic versions of these services. Hence, a dynamic online reconfigurator for our operating system is required.

3.1 From offline TEReCS to online TEReCS

The TEReCS configurator has originally been developed for the offline configuration of operating systems comprising very fine-grained optional components that are customizable at source code level.

We have integrated the configurator into the operating system for online reconfiguration. In conrast to the offline case, the basic idea is to configure only coarse-grained components for the online case. Thus, the overall decision space is more restricted and the time, spent for the selection of the appropriate components and their parameters, can be shortened.

The advantage of TEReCS to define a hierarchy on the valid design space [3] made the reuse of the configurator for the online case possible. For the online case the definition of a hierarchy leads to clusters at level $n + 1$, e. g. for the system hierarchy, memory management, scheduling, communication, etc., of tightly coupled components of the level n.

Tightly coupled components form a cluster, when their pre-configured source code can be compiled. Thus, all inheritance and

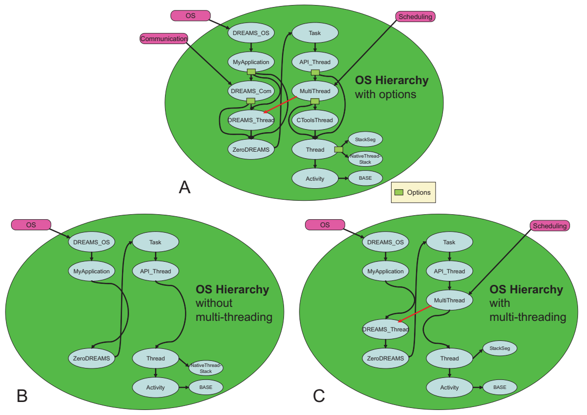

Figure 1: OS cluster with its lower level optional components and two pre-defined configuration examples.

membership relations for the object-oriented design of the RTOS that had been customizable for the offline case must be fixed for the online case. Only calling dependencies to functions of other objects in other clusters are allowed.

Each of these clusters can – more or less – be configured without interference to its neighboring clusters. The dependencies between the clusters can be mapped to *internal primitives*. They become root nodes to the cluster and are triggered/used by other clusters. Thus, independent offline configuration of the clusters for different use cases is possible.

For each of the clusters there exist different pre-configured solutions, which can be implemented. Thus, the level $n+1$ with its predefined solutions defines again a valid design space for TEReCS. The solutions of the clusters made at the lower level become options at the higher level. Normally, the design space at the higher level is smaller than on the lower level, because of the hierarchy.

The online configuration makes use of pre-defined solutions that have been configured offline. Thus, it is up to the online configuration phase to identify the use cases, for which the solutions have been created. This can be seen as a kind of *case-based reasoning* (CBR). For our scenario the identification is simple. The same process, which selects the appropriate options, can be reused. Thus, the complete configuration process works identically to the offline case.

In detail the identification process for the current case and the predefined case is trivial. Basis for the configuration decisions in TEReCS is the use of the *system primitives* that trigger the *use* of services/components. For the predefined case a specific subset of the primitives and specific access parameters is assumed. It is clear that the same assumptions must hold for the current online case and the pre-defined one in order to identify their similarity. In fact, similarity here means identity.

TEReCS especially supports this identification. The same system primitives that have been used to create a pre-defined solution should lead to the selection of that solution component in the higher design space level. This condition must be assured during the specification of the abstract design space for the pre-defined solutions. This problem must be solved by the system expert also offline. This procedure is legal, because TEReCS's main ideology obliges the encapsulation of all expert knowledge in the design space descriptions.

The Figures 1 and 2 sketch an example for two pre-defined cluster options (B+C). The primitives *Scheduling* and *Communication*

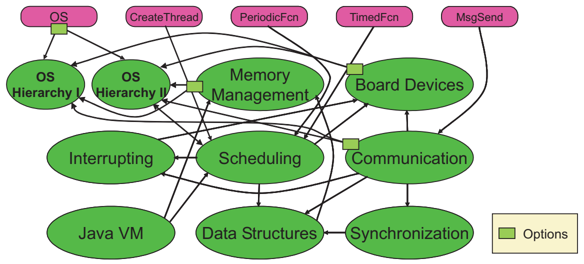

Figure 2: OS design space at 2^{nd} level with integrated options for pre-defined solutions of clusters.

in Figure 1 are used by the equally named clusters from Figure 2. The option B is generated from A, if the primitive *Scheduling* is not used. The option C is generated alternatively. In Figure 2, the pre-generated solutions B+C are included as the *OS Hierarchy Option I* and *II*. Except the cluster *Scheduling* all other clusters can use both options alternately. Only the cluster *Scheduling* explicitly requires the solution of the *OS Hierarchy II*, which supports for multiple threads. If the primitive *CreateThread* will be used, then the cluster *Scheduling* is requested. Thus, the request of primitive *CreateThread* from an application requests for the cluster *Scheduling* to be instantiated. Moreover, this requests for the option of *OS Hierarchy II* to be instantiated instead of the *OS Hierarchy I*.

3.2 Integration of TEReCS and the FRM into the RTOS

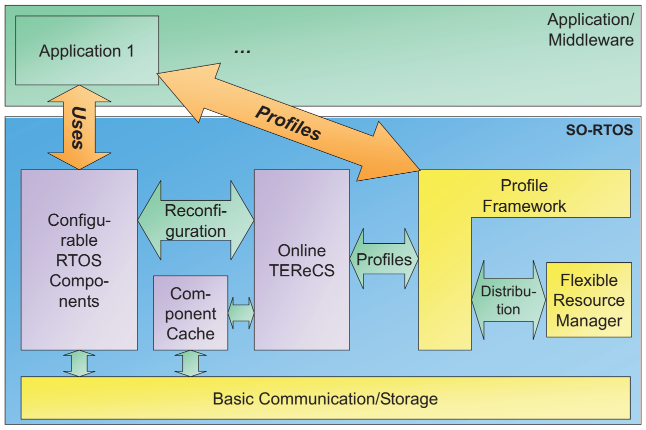

Figure 3: Integration of TEReCS and the FRM framework into the RTOS.

The *Flexible Resource Manager* (FRM) was primary developed to optimize the resource utilization between dynamic applications. Now, additionally recourses of RTOS components are considered by modeling them with our profile framework.

The different options of the RTOS implementation are modeled as different profiles. The resources that are required within the profiles are the *system primitives*. Initially TEReCS holds each primitive. When an application arrives or wants to use a primitive it has to claim for an appropriate profile, which is able to access the primitive. Thus, the FRM forces TEReCS into a profile, where it does not use the primitive. The meaning for TEReCS of holding the access to a primitive is reverse to the meaning of an application: TEReCS holding a primitive means that the primitive is not required and must not be implemented. The other way round, when an application holds a primitive, TEReCS has to provide the primitive's code.

As sketched in Figure 3 the reconfiguration of the RTOS cluster components is completely managed by TEReCS. The reconfiguration options are modeled as optional profiles that are offered by TEReCS and managed by the FRM. Each profile defines exactly, which primitive is used and, which is not used by the applications. Thus, the FRM needs not to distinguish between the RTOS and normal applications. The FRM mediates the *system primitives* (resources) between all the applications and the RTOS. Thus, it handles the competition between the applications and the reconfiguration options of the RTOS. Real-time constrains are respected by modeling the reconfiguration time of the RTOS in the switching conditions of the profiles and the acceptance test in the FRM.

Applications must define all real-time constrains regarding their future resource allocations. Additionally, an application can only allocate resources in the ranges of the profile, which is currently active. With this information the FRM guarantees by courtesy of the acceptance test that all resource allocations can be performed in the maximal assignment delay. The FRM forces TERECS only to deactivate a system service if it can be activated "in time" to provide the resources. This means, that the maximum assignment delay of the applications according to a resource of a system service must be greater than the reconfiguration time to re-activate this system service. Otherwise the system service could not be deactivated.

The online reconfiguration model that has been presented arises some questions concerning its implementation.

The creation of pre-defined solutions for the clusters can be done automatically. For each combination of possible requests or dismissals of *system primitives* and *internal primitives* a configuration is generated. For the optimization and the reduction of the design space of the operating system, a system expert might restrict the combinations of parallel instantiated system primitives to only those ones that make sense in a way that they cover other solutions and – with high probability – are used not simultaneously.

A repository stores all pre-defined solutions of the clusters. A cache will temporarily store the code and description of optional configurations for clusters, in order to speed up the loading of required cluster implementations. The cache might be able to retrieve other configuration's implementations from a background storage (hard disk) or from the network (see Fig. 3).

The technical exchange of two configurations during run-time is based on dynamic online linking. The configuration uniquely identifies a specific pre-configured cluster component. The component's code is attached to the appropriate ELF headers. Additionally, the already instantiated system's code is also described by appropriate ELF headers. Thus, re-linking of new code is possible during run-time by a modified ELF object code linker, which is included into the **Online TEReCS**.

The FRM tries to optimize the system according to the current resource requirements of the components (system services and applications) and the quality information of the profiles. To do this the **Flexible Resource Management** requests the application and TEReCS (which manages the RTOS profiles) to change the current profiles. This results into a reconfiguration of the RTOS and a optimization of the resource usage between the applications and the operating system.

The capabilities for the optimization are exploited by TEReCS during offline generation of the pre-defined cluster configurations for specific use cases. Additionally, the optimization is done always by the FRM based on the quality values.

The FRM includes the definition of quality values per profile. Thus, the FRM can not only reason about the optimality of application profiles, but it can additionally reason about the optimality of the RTOS configuration.

4. VALIDATION

To validate our approach we have implemented the FRM and the Online TEReCS component on top of our RTOS DREAMS and tested the system with an realistic application example from our self-optimizing context. For the purpose of a case study [6] we selected a component of the real-time communication system (RCOS): A reconfigurable ethernet switch.

5. CONCLUSION

In this paper, we have shown the extension of an offline OS configurator to the online case. The combination of the model of the OS reconfiguration options and the management concerning different resource profiles of the applications has been presented. By the integration of TEReCS and the FRM into a RTOS we derived a self-optimizing real-time operating system (SO-RTOS). A simple prototype of our SO-RTOS validate that our approach is functionally working and has a benefit for the executed applications with an increase of the resource utilization. For the future we plan to apply our approach to larger examples.

6. REFERENCES

[1] C. Böke. *Automatic Configuration of Real-Time Operating Systems and Real-Time Communication Systems for Distributed Embedded Applications*. Phd thesis, Faculty of Computer Science, Electrical Engineering, and Mathematics, Paderborn University, Paderborn, Germany, 2003.

[2] S. Burmester, M. Gehrke, H. Giese, and S. Oberthür. Making mechatronic agents resource-aware to enable safe dynamic resource allocation. In *Fourth ACM International Conference on Embedded Software (EMSOFT'2004)*, 27 - 29 September 2004.

[3] R. P. Chivukula, C. Böke, and F. J. Rammig. Customizing the Configuration Process of an Operating System Using Hierarchy and Clustering. In *Proc. of the* 5^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, pages 280–287, Crystal City, VA, USA, 29 April – 1 May 2002. IFIP WG 10.5. ISBN 0-7695-1558-4.

[4] C. Ditze. *Towards Operating System Synthesis*. Phd thesis, Department of Computer Science, Paderborn University, Paderborn, Germany, 1999.

[5] K. Ecker, D. Juedes, L. Welch, D. Chelberg, C. Bruggeman, F. Drews, D. Fleeman, and D. Parrott. An optimization framework for dynamic, distributed real-time systems. *International Parallel and Distributed Processing Symposium (IPDPS03)*, page 111b, April 2003.

[6] B. Griese, S. Oberthür, and M. Porrmann. Component case study of a self-optimizing rcos/rtos system: A reconfigurable network service. In *Proceedings of International Embedded Systems Symposium 2005*, Manaos, Brazil, 15 - 17 August 2005.

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