Concepts for Swarm System Software

Daniel Graff, Jan Richling
Communication and Operating Systems Group
Technische Universität Berlin
10587 Berlin, Germany
Email: {daniel.graff,jan.richling}@tu-berlin.de

Matthias Werner
Operating Systems Group
Chemnitz, University of Technology
09111 Chemnitz, Germany
mwerner@informatik.tu-chemnitz.de

I. Approach

Cyber-physical systems (CPS) tightly interact with their physical environment "*usually with feedback loops where physical processes affect computations and vice versa*" [1]. In the future there will be a plethora of devices [2] implying a variety of programming approaches, user interface concepts and system software, making the handling of these devices and especially the cooperation between different devices even harder. Therefore, it is an established idea to consider the sum of all these devices as *one* emerging system: the swarm [3].

Considering the challenge of dealing with such a swarm from a historical perspective, this situation is very similar to the one regarding general purpose computers in the 50s of the last century. Here, a computer was exclusively used for one application that had full control over all resources while multiple programs were only possible by batch operation. This resulted in rather low utilization of resources and bad response times for many applications. The solution was the introduction of time-sharing operating systems that allowed the operation of several independently developed applications in parallel. Resources were no longer directly controlled by the application, instead, appropriate abstractions of the operating system were used to enable it to manage them in a reasonable manner.

Thus, our approach is to design a swarm operating system that has to take control of resources while enabling several swarm applications in parallel. These applications are developed independently from each other and no longer control hardware resources directly but by means of the swarm operating system. In standard operating systems, the basic concept for this is virtualization both for resources as well as for execution. Therefore, the swarm operating system virtualizes the physical swarm in a way that it becomes several virtual swarms (Figure 1), each dedicated for one swarm application [4]. Internally, it maps virtual to physical resources in a time- and space sharing manner.

This approach raises two main questions: What is an appropriate model to program swarm applications for such an operating system (Section II) and what is a suitable runtime system that is required for managing them (Section III)?

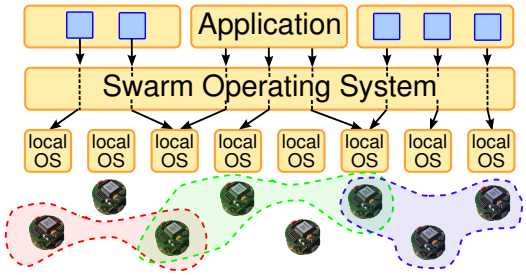

Figure 1. Swarm Operating System

II. Programming Model

Interactions between computational and non-computational elements, as typical for CPS, involve real space and time and, thus, have to be coordinated on the computational side introducing concurrency and distribution which is often error-prone for application developers.

Thus, we argue for an adequate programming abstraction which facilitates application development by introducing a systemic view to system resources which hides concurrency and distribution from the developer while synchronization and coordination become implicit.

Going further, we provide functionality to the developer to specify the programs intention (WHAT to do) rather than to explicitly program every detail (HOW to do) and, thus, reduce complexity. In [5], we presented this paradigm by introducing the concept of *Distributed Active Objects* as a design pattern in order to program such swarm applications.

According to this pattern, programs consist of building blocks that we call *actions* that represent interactions with the physical world. Such an action is comprised of imperative code (encapsulated in a method) describing the impact on the physical world, a spatial-temporal constraint that binds the action to space and time and a result that is the outcome of the action. For instance, taking a picture in the time interval $[t_1, t_2]$ at location X is an action that is constrained in space and time. Such actions are programmed in a sequential manner—without synchronization or coordination—and only the assignment of spatial-temporal constraints induces implicit concurrency of actions.

Furthermore, the programmer specifies high level goals

for such applications that, in order to reach the goal, require a specific composition of actions. Thus, an entire program emerges implicitly only based on a set of basic building blocks (actions) and spatial-temporal constraints. In this case, spatial-temporal constraints are used in order to glue actions together which generates relations among actions. This can be, for instance, synchronous, parallel or dependent execution. The composition itself is not defined by the programmer, but emerges implicit.

III. Execution Model

According to our approach, applications are isolated from each other and behave as if they were using system resources exclusively. Thus, on the system level, we have to guarantee the isolation by coordinating actions in space and time which allows us to efficiently use system resources. For this, we apply the concept of virtualization. Therefore, we plan the entire amount of actions—from all applications—on virtualized resources (exclusive for one application) and map them to physical (shared by all applications).

Before the actual mapping process takes place, an action management is performed in order to efficiently use system resources. The basic idea is to merge actions in order to reduce the resource utilization by using similarities between actions. In this case, different actions on virtual resources given by several applications are mapped to exactly the same action on physical resources.

After the merging phase, the mapping phase tries to find a spatial-temporal mapping from the actions defined on virtual resources to physical. The mappings are dynamic and, thus, may change over time. This allows, for instance, to replace physical movement (of resources) by logical movement (migration of code) in order to reduce energy consumption. However, in order to calculate solutions for the mapping, the following steps have to be performed: a *job scheduling* in space and time and a *path planning* in order to move mobile resources along a spatial-temporal trajectory. We identify 3 kinds of mappings: a **1:1** mapping in which one virtual resource is directly mapped to one physical. In some cases, this is not feasible and, thus, sensor fusion becomes an option inducing a **1:n** mapping. Last, in the case one resource is capable of satisfying the needs of other actions too, an **n:1** mapping is performed.

Finally, a dedicated runtime system is required in order to manage, coordinate, synchronize and execute swarm applications. For this, we propose a service-oriented architecture that consists of local and distributed services. Distributed services are used for global system management, resource allocation and controlling of nodes. The core services of the system are the job scheduler and the path planner which are responsible for planning jobs as well as calculating collision free trajectories. Local services are used for local node management including control of local sensors and actuators, monitoring of code execution (swarm application) and following spatial-temporal trajectories (by accurate engine control).

IV. Conclusion

In this paper, we showed concepts for swarm system software in order to realize a swarm operating system providing a common interface to system resources which mediates between the sum of different devices and swarm applications developed by different programmers. The system guarantees isolation of swarm applications by coordinating access to shared resources (under the assumptions of a given fault model). We have proposed virtual swarms as an abstraction of the physical swarm representing an execution environment for swarm applications enabling multi-program operation. We introduced a programming abstraction enabling a systemic view to virtualized system resources and presented a design pattern to achieve high level functionality facilitating application development for CPS. Finally, we presented an execution model that states how the virtualization and management of swarm applications is realized.

Similar to "Platform as a Service" as part of the cloud computing approach, a swarm operating system might enable "Swarm as a Service" by offering different independent users the possibility to run their swarm applications on large (private or public) swarm systems. This could envision a whole software market for swarm applications.

References

[1] E. A. Lee, "Cyber-Physical Systems – Are Computing Foundations Adequate?" in *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, October 2006.

[2] M. A. Uusitalo, "Global Vision for the Future Wireless World from the WWRF," in *IEEE Vehicular Technology Magazine*, vol. 1, no. 2, 2006, pp. 4–8.

[3] E. A. Lee, J. D. Kubiatowicz, J. M. Rabaey, A. L. Sangiovanni-Vincentelli, S. A. Seshia, J. Wawrzynek, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, R. Jafari, D. Jones, V. Kumar, R. Murray, G. Pappas, A. Rowe, C. M. Sechen, T. S. Rosing, B. Taskar, and D. Wessel, "The TerraSwarm Research Center (TSRC) (A White Paper)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-207, Nov 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-207.html

[4] M. Werner, J. Richling, and G. Mühl, "Two Abstractions for Distributed Systems of Mobile Nodes," in *Seventh International Conference on Information Technology (ITNG 2010)*, 2010, pp. 1219–1220.

[5] D. Graff, J. Richling, T. M. Stupp, and M. Werner, "Distributed Active Objects – A Systemic Approach to Distributed Mobile Applications," in *8th IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, R. Sterrit, Ed. IEEE Computer Society, April 2011, pp. 10–19.