jSwarm: Distributed Coordination in Robot Swarms

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
Email: mwerner@informatik.tu-chemnitz.de

***Abstract*—We present a runtime system for swarms of mobile robots that manages distributed resources and provides a common programming interface for distributed swarm applications. The programming abstraction follows a systemic view and allows to specify the spatial-temporal behavior of applications. The runtime system analyzes application code, creates a dependency graph, extracts spatial-temporal actions and uses code rewriting in order to realize a distributed execution coordinated in space and time.**

I. Introduction

Allowing a glance in the past, technical evolution has shown that devices have become more powerful, use less energy, are equipped with a variety of sensors and actuators and have become strongly interconnected and mobile. This has led to the emergence of cyber-physical systems (CPS) that tightly interact with their physical environment "*usually with feedback loops where physical processes affect computations and vice versa*" [1]. Already in the near future, there will be an enormous amount of wireless devices [2] ranging from deeply embedded sensors and actuators over intelligent gadgets to fully autonomously acting robots forming sensing and actuating platforms featuring a variety of hardware and software, different programming approaches and system interfaces making, especially, the cooperation and coordination a challenging task. Thus, our approach is to consider all these devices as *one* emerging system (the swarm) and build a distributed swarm runtime system on top of it that hides heterogeneity and diversity, provides a common interface to the outside and decouples swarm applications and their execution in space and time. For this, we provide a programming abstraction featuring a systemic view to system resources together with the possibility of attaching spatial-temporal constraints to components of the application.

The remainder of the paper is structured as follows: Section II sketches the approach and describes the programming model for swarm applications that can be bound to space and time. Section III shows an architecture of the swarm runtime system that we currently develop according to the aforementioned approach. Finally, Section IV summarizes related work and Section V concludes the paper.

II. Approach

Similar to the process concept (virtualization of the processor) in order to allow multi-program operation and a higher utilization of the processor which is a standard in todays operating systems, we propose the virtualization of the swarm

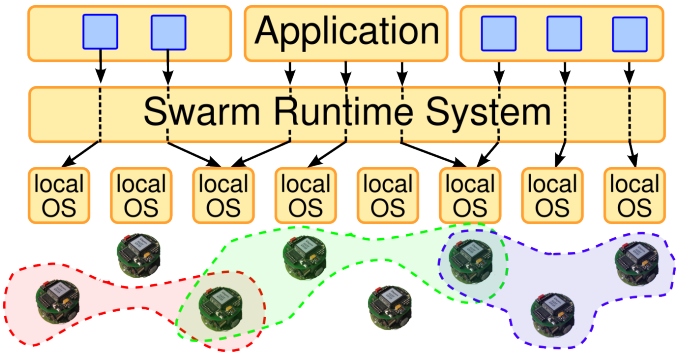

Fig. 1. Multi-program operation enabled by the use of virtual swarms (time-varying mapping of virtual resources to physical resources) which provides isolated execution

consisting of mobile devices such as robots. In contrast to approaches where only one program is executed and allocates the entire swarm, we allow multiple parallel swarm applications by executing each application on a separate virtual swarm which is a subset of the real swarm as depicted in Figure 1. This is fully transparent for applications and isolates execution from each other. Applications do not access resources exclusively, but in a time sharing manner coordinated by the runtime system. According to this approach, applications define operations on virtual resources that are mapped to physical resources. A virtual swarm then is defined as a time-varying mapping of virtual resources to physical resources.

Experience from distributed systems shows that programming concurrent, parallel or even distributed applications is already error-prone. Incorporating real-time and space as a necessity for CPS makes things worse.

In [3], we have presented a programming abstraction based on *Distributed Active Objects* that follows a systemic approach. We call an interaction with the physical world by means of sensors and actuators an *action* that can be restricted in space and time. Actions are defined on virtual resources. Since the system provides a systemic view to system resources, programmers do not have to cope with distribution or concurrency, they are able to access them in a local manner. Due to the introduction of spatial-temporal constraints that can be attached to actions, programmers implement applications sequentially that will become distributed and concurrent implicitly at runtime.

With this concept, we are able to define high level goals in an easy way. For instance, consider the example of a 3-sided observation of an entity of the real world as depicted in

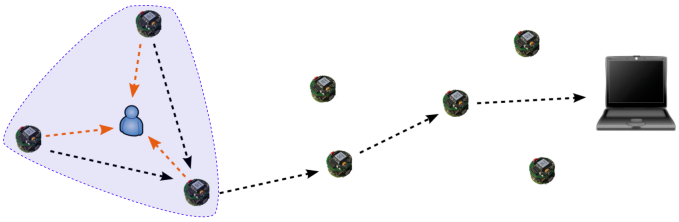

Fig. 2. 3-sided observation: neither quantitative nor qualitative aspects are specified; the formation will emerge implicitly based on application constraints

Figure 2. Here the idea is to take pictures from three different sides at the same point in time in order to compute a 3D reconstruction afterwards. Doing this manually incorporates selecting a subset of robots featuring a camera resource, positioning the resources accordingly, synchronize under each other, take the pictures and finally collect the resulting images. According to our approach, we abstract from physical resources and provide a systemic view in which virtual resources can be constrained in space and time. The following shows how spatial constraints can be expressed:

$$\begin{aligned} &\forall i \in \{0,1,2\}\ \exists j = (i+1) \mod 3 \mid \\ &\quad (dist(C_i, C_j) == D_1 \ \wedge \quad // \ D_1 \in [..] \\ &\quad dist(C_i, X\ ) == D_2) \quad // \ D_2 \in [..] \end{aligned}$$

While the X value is the position of the entity that shall be observed, the C_i parameters indicate positions for different camera resources that have to be computed regarding X under the restriction that each pair of cameras shall have a distance of D_1 and each camera shall have a distance D_2 to X. This expression results in the formation of an equilateral triangle surrounding the observed entity. In addition the programmer is able to define a temporal condition such that all pictures are taken at the same point in time indicated by t_i:

$$\forall i \in \{0,1,2\}\ \exists t_i == T \quad // \ T \in [..]$$

Each position C_i is associated with a corresponding camera object that is used for taking the pictures. Based on the constraints, the programmer is now able to simply invoke the `takePic()` method of the camera objects, get the resulting image and perform the reconstruction afterwards in a sequential manner as shown in Listing 1. All coordination, synchronization, selection and movement of the distributed resources is fully transparent to the programmer.

```
1 public 3dImage do3dReconstruction() {
2   Pic a = c1.takePic(); // C1
3   Pic b = c2.takePic(); // C2
4   Pic c = c3.takePic(); // C3
5   return reconstruct3d(a,b,c);
6 }
```

Listing 1. Code written by the programmer

III. Runtime System

In order to execute swarm applications developed according to the programming model, we present *jSwarm* as a distributed runtime system which follows a service-oriented architecture approach. The system consists of distributed (global) and local service as depicted in Figure 3. Global services are used for system management while local services are used for local node management. The system can be operated in two modes: In static mode all applications are perfectly known in advance and offline scheduling is applied in order to plan application in space and time. In dynamic mode, applications can be started during runtime requiring an online scheduler.

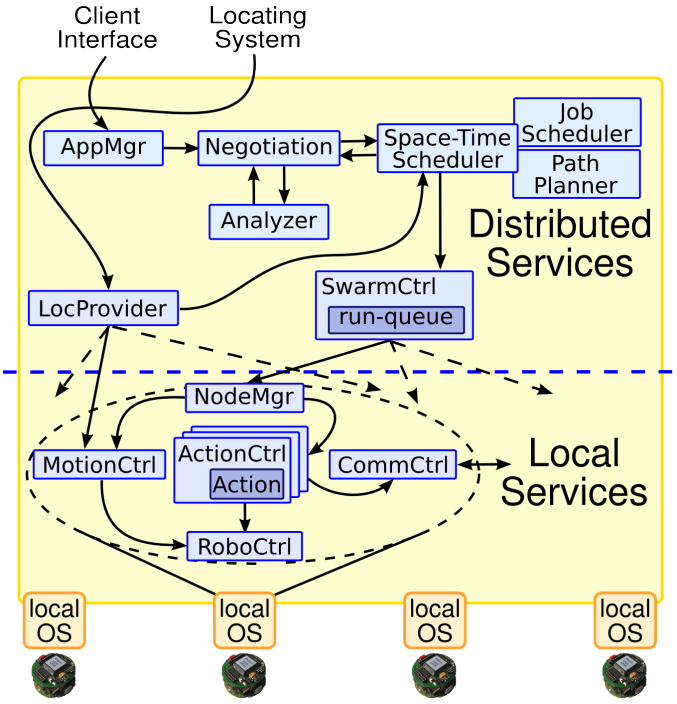

Fig. 3. Architecture of the runtime system which consists of local (executed on each device for node management) and distributed services (executed as system-wide singleton for system management)

The distributed services are globally reachable system services that are responsible for managing the system. During system startup a leader election algorithm is used to determine where the services shall be executed. Each service is replicated as backup service in case of failures. Nodes autonomously find each other using a discovery service that performs UDP broadcasts. As soon as other nodes have been found, reliable TCP connections are established to the explored nodes.

In order to deploy and start a new application it is possible to connect to the swarm using the client interface. After the code has been fully loaded into the swarm, the *Analyzer* parses the applications code, extracts all space-time constraints and constructs a dependency graph of the application as depicted in Figure 4. In case of the 3-sided observation (Listing 1), the code contains 3 actions attached with space-time constraints in order to take the 3 pictures. For simplicity, it is assumed that the application consists of two sequential parts (one before taking the picture and one afterwards). The resulting graph, thus, shows relation between the 3 actions for taking the pictures (having the same timing constraint, but different spatial constraints) and, thus, is executed in parallel while computing the 3D reconstruction has to be done afterwards.

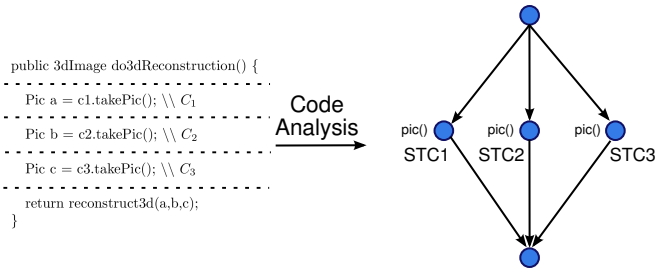

Fig. 4. Code analysis: generating a precedence graph based on spatial-temporal application constraints

After the code analysis, a code generator cuts out spatial-temporal actions and hands them over (together with the spatial-temporal constraints) to the *SpaceTime-Scheduler* which consists of a job scheduler (plans actions in space and time) and a path planner (computes spatial-temporal trajectories for moving resources). The scheduler has to guarantee correctness of a schedule, i.e., if new jobs arrive and shall be scheduled, an existing correct schedule (containing already scheduled actions) is always transformed into another correct schedule. The scheduler uses the spatial-temporal constraints as input values for the computation of the schedule and evaluates their operators as given in Section II with respect to system constraints (inaccuracies of locating system, inaccuracies of sensors and actuators, e.g., engine control). Hence, the "==" operator involves fuzziness to a certain degree that depends on current system capabilities. In [4], we presented an approach for modeling offline group scheduling problems in space and time.

The scheduler uses an existing schedule (containing already scheduled actions) and tries to update it. If the scheduler is not able to find a schedule, the application is semi-rejected. It is still possible to do a renegotiation with the application to shrink resource usage. Finally, if the space-time scheduler has successfully planned the actions, the *SwarmCtrl* updates its run-queue accordingly. The SwarmCtrl is responsible for assigning planned jobs (either spatial-temporal actions or trajectories) to the respective nodes.

```
1  private Pic a,b,c;
2  public 3dImage do3dReconstruction() {
3    wait();
4    return reconstruct3d(a,b,c);
5  }
6  public receive(Message m) {
7    // extract picture from message
8    // and assign to respective variable
9    if (a && b && c) {
10     notify();
11   }
12 }
```

Listing 2. Code generated by the system

The remaining method body is then modified as depicted in Listing 2. What happens now is that as soon as the code of the modified application gets executed and an invocation of `do3dReconstruction` is performed, the executing thread will be blocked in `wait()` waiting for the pictures to arrive. It is not necessary (for the application itself) to request the pictures since the scheduler has already planned those actions in space and time. Each time when a picture has been taken, it will be send back to the waiting thread causing an invocation of `receive(..)`. As soon as all three pictures have arrived, `notify` causes the blocked thread to be woken up which will finally continue and call `reconstruct3d(..)`.

In order to guarantee the proper execution, each node owns the same set of local services. The *NodeManager* is the central element and is responsible for local node management. It receives new jobs (spatial-temporal actions or trajectories) from the *SwarmCtrl* and puts them in its own queue sorted according to time stamps (when to execute). Depending on the jobs, it informs either the *MotionCtrl* or *ActionCtrl* which both have their own thread of control. In case that a resource has to be moved, the *MotionCtrl* is responsible for moving along a spatial-temporal trajectory by engine control. The robots hardware can be accessed using *RoboCtrl*. In case of a spatial-temporal action, the code is handed over to the *ActionCtrl*—a wrapper for actions—for execution. Finally, the *CommCtrl* is used in order to send return values of actions back to the calling thread.

IV. Related Work

In [5], SwarmOS is presented as a mediation layer between applications and distributed resources such as sensors, actuators, storage and computing. It has to cope with distribution, heterogeneous and shared resources as well as dynamic situations (mobility, connectivity, ..) while providing context awareness. In contrast to our research, SwarmOS does not feature (or is not intended) to support autonomous resource movement that is based on spatial-temporal trajectories that are automatically computed from the system based on a resource usage of applications. The same holds for MagnetOS [6] which provides to split an application into components each of which will be dynamically assigned to several executing nodes (targeting ad-hoc and sensor networks).

In [7] an approach for dynamic task assignment in robot swarms based on swarm algorithms with stochastic elements is presented. It supports groups of nodes (especially robots) that collectively execute applications without the need to program the nodes separately, i.e., it provides location transparency within the group. It is intended to support applications in real-space. However, it is designed to support swarm algorithms with stochastic elements. It does not provide a notation of real time or space.

The Symbrion and Replicator projects [8] consist of super-large-scale swarms of robots that are based on bio-inspired approaches featuring self-X properties. The systems are highly dynamic and so, if advantageous, the robots can aggregate into a symbiotic organism that is, in this form, better suited to solve a task in the current situation while sharing resources such as energy.

There are different kinds of programming abstractions for distributed, concurrent and parallel systems as well as for sensor networks. Detailed surveys are provided in [9], [10]. Following a holistic approach, nesC [11] which is an extension to C is a programming language for deeply networked systems which was created for TinyOS. Programs are built out of components that have internal concurrency. While nesC is a node-level language (code is written for an individual node), Pleiades [12] provides an abstraction to implement a central

program that has access to the entire network (also known as macroprogramming [13]). SpatialViews [14] is an extension to Java which allow to define virtual networks that are mapped to physical nodes according to their physical location and the services they provide. Execution is distributed among the nodes in the virtual network performed by code migration. Furthermore, it is possible to constrain execution based on timing restrictions.

In [15] a programming and execution environment for micro-aerial vehicle swarms is presented. Applications are a composition of low level drone behaviors and high level goals that are submitted by a user for execution on the swarm. While this approach is similar to ours, we further provide to use application constraints in order to define spatial and temporal conditions (synchrony, (in)-dependence) in a relative or absolute manner for a set of resources.

V. Conclusion

In this paper, we presented an approach for a swarm runtime system as a mediation layer between swarm applications and distributed resources such as sensors and actuators. The runtime system hides heterogeneity and diversity of hardware, software and system interfaces and provides a clear defined interface for applications. In addition, we provide context awareness for applications while guaranteeing location and distribution transparency.

After the system has received a new application for execution, it will be analyzed and according to the spatial-temporal actions a dependency graph is computed. Afterwards, the actions are extracted and code rewriting is used in order to realize a distributed execution coordinated in space and time. If necessary, spatial-temporal trajectories are calculated in order to move resources. Finally, actions and trajectories are scheduled to nodes in the system for execution.

According to the programming model, we allow the programmer to implement in a sequential manner without the need to use neither coordination nor synchronization and to bind parts of applications to space and time while resource movement, scheduling and mapping is fully transparent. The swarm runtime system supports to execute multiple independently developed applications that can be executed in parallel by guaranteeing isolation achieved by using the concept of virtual swarms.

For performing experiments, we have set up a swarm lab consisting of 40 mobile robots equipped with a 400 MHz ARM9 CPU and 64 MB SDRAM and 24 stationary boards. equipped with a 180 MHz ARM9 CPU and 64 MB SDRAM. We use an external locating system for indoor tracking of the robots based on a Microsoft Kinect [16]. We have implemented basic functionality for the runtime system such as node detection, communication and a simple way of executing swarm application. Using a simulator, we tested the calculation and following of trajectories by fine-grained engine control. Currently, we are working on testing the algorithms in our swarm lab with the real hardware.

Since mobile devices have a limitation of resources, a major research field still remains to further investigate suitable heuristics for online scheduling.

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] D. Graff, J. Richling, T. M. Stupp *et al.*, "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.

[4] D. Graff, J. Richling, and M. Werner, "Modeling Group Scheduling Problems in Space and Time by Timed Petri Nets," *Fundamenta Informaticae*, vol. 122, no. 4, pp. 297–313, January 2013.

[5] E. A. Lee, J. D. Kubiatowicz, J. M. Rabaey *et al.*, "The TerraSwarm Research Center (TSRC) (A White Paper)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-207, Nov 2012.

[6] R. Barr, J. C. Bicket, D. S. Dantas *et al.*, "On the Need for System-Level Support for Ad hoc and Sensor Networks," *Operating System Review*, vol. 36, pp. 1–5, 2002.

[7] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms," in *Robotics: Science and Systems Conference*, Cambridge, MA, USA, 2005.

[8] S. Kernbach, E. Meister, F. Schlachter *et al.*, "Symbiotic robot organisms: REPLICATOR and SYMBRION projects," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, ser. PerMIS '08. New York, NY, USA: ACM, 2008, pp. 62–69.

[9] R. Sugihara and R. K. Gupta, "Programming Models for Sensor Networks: A Survey," *ACM Trans. Sen. Netw.*, vol. 4, no. 2, pp. 8:1–8:29, Apr. 2008. [Online]. Available: http://doi.acm.org/10.1145/1340771.1340774

[10] L. Mottola and G. P. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art," *ACM Comput. Surv.*, vol. 43, no. 3, pp. 19:1–19:51, Apr. 2011. [Online]. Available: http://doi.acm.org/10.1145/1922649.1922656

[11] D. Gay, P. Levis, R. von Behren *et al.*, "The nesC language: A holistic approach to networked embedded systems," *SIGPLAN Not.*, vol. 38, no. 5, pp. 1–11, May 2003.

[12] N. Kothari, R. Gummadi, T. Millstein *et al.*, "Reliable and efficient programming abstractions for wireless sensor networks," *SIGPLAN Not.*, vol. 42, no. 6, pp. 200–210, Jun. 2007.

[13] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 3–3.

[14] Y. Ni, U. Kremer, A. Stere *et al.*, "Programming ad-hoc networks of mobile and resource-constrained devices," *SIGPLAN Not.*, vol. 40, no. 6, pp. 249–260, Jun. 2005.

[15] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh, "Programming Micro-aerial Vehicle Swarms with Karma," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11. New York, NY, USA: ACM, 2011, pp. 121–134. [Online]. Available: http://doi.acm.org/10.1145/2070942.2070956

[16] M. Haustein, A. Löscher, and M. Werner, "Adaptive Objektlokalisierung durch Tiefenbildanalyse mittels einer Kinect-Kamera," in *Ortsbezogene Anwendungen und Dienste, 9. Fachgespräch der GI FG KuVS*, 2013, pp. 109–118.