A Top-Down Approach for a Generic Programming Abstraction in Real Space-Time

Martin Däumler · Dirk Müller · Matthias Werner

Abstract Today's widely used programming approach for mobile distributed systems, e.g., swarms, is bottom-up. I.e., the programmer has to be aware of the system's distribution. Such a kind of programming corrupts the principle *Separation of Concerns* and turns out to be complicated. This article proposes a top-down approach for the programming of mobile distributed systems. It should be incorporated in a new operating system to be developed by our research group [17] [18].

Classical distributed systems often are intended to hide their distribution from the user but not stringently from the programmer. Moreover, most applications don't consider the executing system's location and mobility, respectively. However, state-of-the-art mobile distributed systems' applications are widely based on location and motion data. So, the programming approach of classical distributed systems which abstracts from location and motion might no longer be convenient. We suggest raising the level of abstraction in order to hide the system's distribution from the user and programmer, in contrast to the bottom-up programming approach of, e.g., swarms. This distribution transparency within the executing system is intended to be combined with location/-motion awareness within the application so that the requirements of modern applications can be met. The programming of such a distributed mobile system will be separated from the complex and error-prone application partitioning and assignment a bottom-up approach would impose. This offers further benefits like scalability and robustness with regard to scheduling of sub-activities and sub-systems, respectively. We promote the use of spatiotemporal constraints to realize such a top-down approach. These constraints will be introduced and explained using two examples.

Martin Däumler · Dirk Müller · Matthias Werner
Operating Systems Group
Chemnitz University of Technology
09111 Chemnitz, Germany
E-mail: mdae|dirkm|mwerner@cs.tu-chemnitz.de

Keywords real space-time · scheduling · spatiotemporal constraints · constraint programming

1 Introduction

Networking, *miniaturization*, *embedding*, and *mobility* are important trends in the ICT field. Related to that, there is an increasing number of distributed and mobile applications. Many of them are characterized by specific communication patterns like client/server or peer-to-peer, and the explicit use of dedicated protocols. Thus, such a distributed application can be seen as a group of nodes where the communication and cooperation has to be specified *explicitly*, called *node view*.

By comparison, a *system view* with *transparency* regarding several aspects of distribution can raise the level of abstraction. The required communication and cooperation between the nodes is performed *implicitly* by some middleware or operating system supporting the abstractions. Thus, the programmer (and user) don't need to care about certain distribution issues like the executing system comprising several sub-systems. The distributed system can be used and programmed as a one-piece system. However, applications that have to be aware of certain locations/motions are usually designed with the node view in mind. So, we will ascertain how to support a system view of a distributed system for programmer and user while keeping motion or location awareness within the application to be executed (and also for programmer/user). This is the objective of the distributed operating system to be developed *FlockOS – **F**ederation of **l**inked **o**bjects with **c**ommon tas**k**s **O**perating **S**ystem*.

1.1 Transparency and Awareness

The need to consider certain properties like the executing system's distribution could make programming (more) diffi-

cult. Hiding such properties from the programmer and user is called *transparency* and could facilitate programming/using the system. In contrast, *awareness* explicitly considers certain issues, e.g., location and motion data as input for state-of-the-art mobile applications.

In computer science, transparency can be seen as a black box approach with encapsulation whereas awareness follows a white box approach. As a first conclusion, both terms stand in a trade-off relationship to each other. However, applying these concepts to different entities could help developing powerful applications. In order to delimit our concept, we specify which kind of transparency and awareness, respectively, should be supported: *Distribution transparency* describing a system abstraction from locations *within* itself in order to realize the system view to both user and programmer. In contrast, *location and motion awareness* are essential features of the *application*. Processing motion or location data could be done passively or actively, i.e., the application not only reacts to given input but rather affects location and motion data of related objects or itself (in-) directly. As a consequence, some of the intended applications require their executing system to be mobile. This does not conflict with the introduced distribution transparency within the system because it should be handled as one system even if it is mobile. Thus, FlockOS has to have means to provide motion and location awareness to the application and an abstraction that hides executing system's distributed character from user and programmer.

1.2 Real Space-Time

Numerous existing embedded systems are real-time systems where the correctness of an application does not only depend on a computation result but also on meeting given timing constraints. These timing constraints are typically expressed by deadlines. Real-time requires an appropriate scheduling that takes timing constraints into account.

Scheduling singly based on time properties ignoring the presence of space may not be appropriate for mobile distributed systems. Space has to be taken into account for applying our approach of location and motion awareness, since the intended applications might be subject to spatial restrictions. Thus, the integration of space and time leads to the concept of *Real Space-Time*. Only a *holistic* view at time *and* space can provide correct results. A reductionistic view taking into account just one of them is possible only in stationary, i.e., non-moving systems.

1.3 Applications

With the spreading of mobile systems, there is a variety of applications that can be simplified or even just realized following our approach. Below, some example scenarios shall give an impression:

- Modern cartography bases on systems of cooperating satellites. Controlling them could be simplified enormously applying such a systemic point of view.
- Traffic jams waste time, resources and money. The approach of *Hovering Data Clouds (HDCs)* [16] could help limiting them. A decentralized, flexible distributed system shall be constructed.
- Cooperating robots, e.g., robot soccer, could take advantage of a system view.

A more detailed discussion of possible applications can be found in [18].

The remainder of this paper is structured as follows: Section 2 discusses our top-down approach for mobile distributed systems in detail. Section 3 provides information about the suggested programming abstraction for motion aware applications. The use of this programming abstraction is demonstrated by two examples in Section 4. Section 5 discusses related work and Section 6 presents our conclusions.

2 The Top-Down Approach

Section 1 introduced the concepts of motion awareness within the application and distribution transparency within the executing system. The latter is an abstraction of the underlying application executing system and enables a system view to user and programmer. It allows programming applications for distributed mobile systems as if the systems comprises only one piece. A side effect of this concept is to hide further characteristics of distributed systems. [8] distinguishes between different kinds of distribution transparency. So, we use this term as a synonym for a subset of it. Which concrete kind(s) are realized often depends on the application's requirements. For example, application migration might be or not be an explicit part of the application. Imagine the observation of a certain area and the observing system comprises several sub-systems that alternate observing the area. According to the system view, the user notices "the system" observing the area and not "the system's sub-systems". In that case, the application's migration from one sub-system to another that continues observing should be handled implicitly. In contrast, an explicit migration may be described generically, i.e., how the application can be influenced by the executing system in order to change the application's identity. For example, a ball in a robot soccer game represents the application and a description how to pass the ball to another soccer player describes the application migration explicitly. The second case is not contrary to the system view as both views have in common that programmer and user are disburdened from specifying how an application is partitioned into several sub-activities and how they are assigned to the

sub-systems, i.e., how they are coordinated and scheduled, respectively. S/he rather programs and uses a virtualized executing system. The concept of system view realized by distribution transparency also allows programming generic applications. A generic application description could be used to describe a class of applications that do not refer to a particular executing system.

In contrast to the programming approach used in most existing swarm (operating system) projects like [7], we want to abandon the bottom-up approach and to realize a top-down programming approach that bases on the system view, i.e., the mobile distributed executing system is programmed as a whole and not with its sub-systems in mind. Our programming abstraction's innovative concept is to support this top-down approach and to integrate it into a new operating system that supports location and motion aware applications. That is, the application's partitioning and the sub-systems' coordination should be shifted to a lower system level, i.e., the operating system, where it can be automated and possibly verified. If that automatization works correctly, it allows using systems that comprise a number of sub-systems whose coordination could not be handled manually. If following a bottom-up approach, a programmer would have to account for sub-system failure manually which would raise the code's complexity and error-proneness dramatically. In our top-down approach, this is intended to be handled by the operating system automatically. The fields of research towards the development of our operating system for distributed mobile systems are already pointed out by [18]:

- Programming Abstraction
- Real Space-Time Scheduling
- Correctness Reasoning

This paper is focused on introducing a suitable programming abstraction for applications in real space-time and particularly on underlining the genericness and flexibility.

3 Programming Abstraction

In this section, we will introduce *spatiotemporal constraints* as generic programming abstraction that allows realizing the top-down programming approach while providing motion awareness within the the application which is beneficial for implementing applications in real space-time where, besides temporal restrictions like in real-time systems, spatial constraints have to be met as well.

Constraints are suitable to express incomplete information and relationships between different objects. E.g., a linear system of equations can be regarded as arithmetic linear constraints [3]. Resolving them is done by a so-called constraint resolver. It might resolve the linear system of equations by using the Gauss elimination. Constraint programming allows solving combinational problems, e.g., scheduling [9], [5], based on incomplete or imprecise information.

Spatiotemporal constraints, i.e., constraints that consider space *and* time information, are intended to extend the imperative part of application code in order to introduce the awareness of restrictions in space, time and motion. A feasible application schedule has to adhere to these given constraints. We consider that spatiotemporal constraints are a suitable programming abstraction because the top-down programming relies on the system view which hides details from the programmer. These have to be (re-) constructed at runtime according to the given constraints. In order to confirm the understanding of how spatiotemporal constraints allow realizing the top-down approach and facilitate motion aware generic programming in real space-time, we distinguish three classes of spatiotemporal constraints:

- Application Constraints
- System Constraints
- Runtime Constraints

Application constraints are given by the application programmer and extend the usual imperative code. They express conditions and restrictions a certain application resource is subject to. Imagine the observation of a certain area on earth by several satellites. The application's execution is bound to a spatial area. The resource this spatial constraint has to be assigned to is a satellite's observation system like a camera. It is available for the application's scope when it is inside the specified spatial area. So, the observation system has to be imposed with a spatial constraint that represents that area. Besides spatial constraints, temporal or motional constraints like a maximum execution time or that the system's velocity that has to be adjusted to a related object's one are imaginable. A generic application description imposes to use generic (constraint) parameters instead of explicit values. That is, some input like a concrete area to be observed may be given later. This provides the possibility that a programmer describes applications having the same intention generically. More abstractly, the applications' executing system can be considered as a virtualized executing system the programmer has not to be aware of its detailed physical properties.

While the application constraints are subject to the application's description which abstracts from the executing system, we suggest using *system constraints* in order to describe the executing system's capabilities. This adheres to the system view because they are intended to specify the motion capabilities and restrictions of an individual sub-system like a maximum velocity or a description of possible motions in space-time. The distinction between application and system constraints allows programming different applications for one system and vice versa while re-using application code. For example, it is necessary to describe a satellite's orbit in order to allow appropriate scheduling.

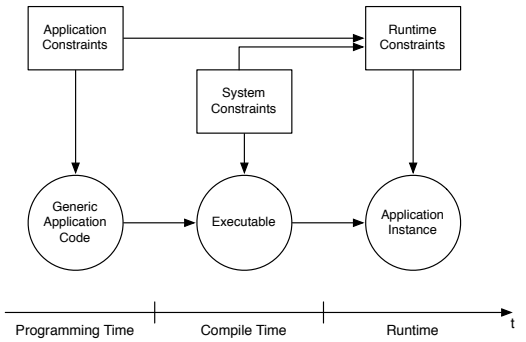

Fig. 1 Different constraint types and their validity within an application's life circle.

So-called *runtime constraints*, which represent runtime information, are used to instantiate all constraints in order to allow a scheduling algorithm to derive a schedule from a generically programmed but later instantiated application. Examples for runtime constraints are the total number of available sub-systems (if this is not static), their initial positions or explicit spatial and temporal restrictions that arise from the current situation. That implies that runtime constraints might be of dynamic nature, i.e., they are allowed to change during execution. On the one hand, a generic application description including numerous runtime constraints increases flexibility and it could be used in a number of different systems. On the other hand, this might increase the solution space for the constraint resolver. So, the distinction between runtime and non-runtime constraints points out to be reasonable in terms of efficient scheduling. Staged scheduling at compile time when only application and system constraints are known might decrease the solution space and so reduce scheduling overhead at runtime. This staged scheduling approach respectively the constraints' validities in an application's life circle are illustrated by Figure 1.

4 Examples

4.1 Problem

In this section, we demonstrate using spatiotemporal constraints as programming abstraction for applications in real space-time. We give two examples that adhere to the same application but different executing systems in order to particularly underline the approach's genericness. The examples help understanding the constraints' intention. We do not focus on the constraint resolving, i.e., how a certain schedule is derived from the application code extended with these constraints. For the ease of understanding, the constraints are given in a mathematical fashion. Further consideration about a suitable language-based programming interface and a stepwise constraint resolving will be published elsewhere. The example application is engaged in observing a certain area. In C-like pseudocode, its imperative programming language part can be given as:

```
1  void* observe_area(int obsystem, int num) {
2
3    image* total = malloc(num*sizeof(image));
4    image frac;
5
6    while( num > 0 ) {
7
8      read(obsystem, &frac, sizeof(image));
9      total[num--] = frac;
10   }
11   return (void*) total;
12 }
```

Listing 1 Pseudocode of the observation application

image represents a data structure for image data. The variable *total* accumulates the pictures taken by the observation system while executing the application. *frac* contains the current taken picture. The *read* operation uses *obsystem* as file descriptor that refers to the observation system, e.g., a camera. This descriptor represents a resource that is required to execute the application. It is available, i.e., the read operation succeeds, iff the application's identity is inside the area to be observed. When the resource is not available, the read operation is assumed not to return, i.e., it is a synchronous operation. In order to realize this resource's availability, a spatial constraint has to be attached to *obsystem*. Application migration to the next sub-system is done implicitly, i.e., it does not need to be considered at application level. However, it implies that the application state, e.g., at least *total* and *num*, has to be submitted to the sub-system that executes the application next. We abstract from the concrete realization and assume application migration is reliable. That application code prescinds from the executing system and therefore adheres to the introduced system view.

4.2 General Conventions and Assumptions

Before starting, several general assumptions and naming conventions are given:

1. The application executing system's sub-systems obj_i are elements of a set of resources R.
2. Each obj_i's location respectively motion is represented by a space-time trajectory (STT) $stt_{obj_i}^{[t_0;t_1]}(t)$. As a motion could be partitioned into several sub-motions, each sub-motion's STT is valid only for a certain time interval. The expression $stt_{obj_i}^{[t_0;t_1]}(t)$ means that obj_i's STT is valid only during the interval $[t_0 \leq t \leq t_1]$.
3. The application identity's location at time t is denoted by $stt_{cur}(t)$, i.e., it is a placeholder for the virtualized executing system the programmer refers to.
4. The spatial area to be observed is represented as a generic closure space $cs(t)$ because its geometric form and in

which coordinate system it is represented is not known at programming time. It represents a set of space-time points.

5. The expression $sst_{cur}(t) \cap cs(t)$ determines if the application, respectively its current identity, is inside the spatial area denoted by $cs(t)$ at time t. If the expression returns the empty set, the application is not inside $cs(t)$. If it is not the empty set, the application must be inside $cs(t)$.
6. In order to determine if the resource *obsystem* is available, it is assumed that a value equal to 1 implies that it is available and a value not equal to 1 that it is not available.
7. Each $obj_i \in R$ has an observation system. The expression $obsystem^{on}_{obj_i}(t_0)$ denotes that obj_i switches on its observation system at time t_0. Analogously, a sub-system switches off its observation system at t_1 expressed by $obsystem^{off}_{obj_i}(t_1)$.
8. When the *read* operation in line 7 from listing 1 is being executed successfully, i.e., the resource *obsystem* is available on the application's identity, this object's observation system is switched on automatically. If the application sleeps or the resource is not available, the observation system is switched off.
9. Each $obj_i \in R$ has a data sending system. $send_{obj_i}(t)$ implies that obj_i sends the application's state to the next application executing sub-system at time t. The communication is assumed to be reliable.

4.3 Satellite Observation

4.3.1 Specific Assumption

In this example, the area observation is done by several satellites on fixed non-geostationary orbits, like illustrated by Figure 2. The specific assumption in this example is that the space-time is restricted to the two-dimensional case, i.e., one polar dimension, the angle, and one time dimension.

4.3.2 Constraints

The following spatiotemporal constraints describe the application's non-imperative part, the satellite system and the runtime parameter. Application constraints are denoted by *aCk*, system constraints by *sCk* and runtime constraints by *rCk* where *k* is the number of the specific constraint.

The application's resource *obsystem* is available iff the application's virtualized executing system $sst_{cur}(t)$'s current position is inside the spatial area $cs(t)$ to be observed. This approach allows specifying the application constraints once for several area observation applications and underlines its genericness.

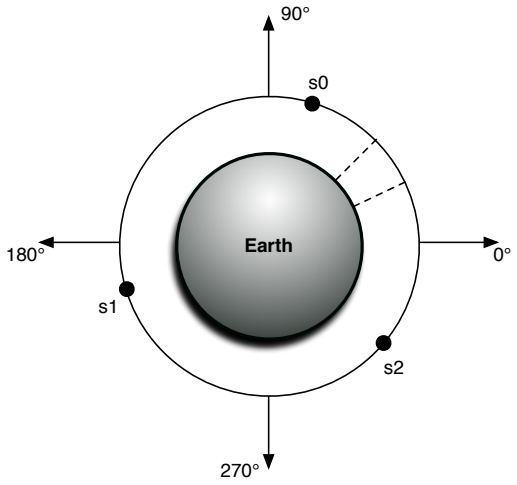

Fig. 2 Three satellites that have to observe the area on the earth's surface marked by the dashed lines.

*aC*1: The application's resource *obsystem* is available iff the application's identity is inside the area to be observed:

$$\forall t : stt_{cur}(t) \cap cs(t) \neq \emptyset \Longleftrightarrow obsystem = 1$$

The system constraints are used to describe the executing system.

*sC*1: There are n satellites which denote the number of available resources. This constraint preserves the distribution transparency because it is given in a generic fashion:

$$\exists n : R = \{s_0, s_1, \ldots, s_{n-1}\}$$

*sC*2: Motion awareness and distribution transparency are preserved by generically describing the satellites' motions. Each satellite s_i's current position at time t respectively motion is described by an STT $stt_{s_i}^{[t_0;t_1]}(t)$, given an initial angle $\sigma_{s_i}^s$ at a starting time t_s and an angular velocity ω_{s_i}:

$$\forall s_i \in R; \forall t; \exists t_s; \exists \omega_{s_i} :$$
$$stt_{s_i}^{[t_s;\infty]}(t) = \left(\omega_{s_i} \cdot (t - t_s) + \sigma_{s_i}^s\right) \mod 360^\circ$$

Runtime constraints can be given by the user, e.g., which spatial area has to be observed, but they also could be determined automatically by the satellite system itself, e.g., the number of available satellites and their initial positions. They instantiate the generic descriptions of the application and the executing system by setting in explicit values as far as possible.

*rC*1: A satellite s_i is inside or outside $cs(t)$ at time t when its current position $sst_{s_i}^{[t_0,t_1]}(t)$ is inside or outside an orbit sector denoted by $\sigma_0(t)$ and $\sigma_1(t)$ that is specified

by the user. This instantiates the application constraint $aC1$:

$$\forall s_i \in R; \forall t, t_0, t_1; \exists \sigma_0(t), \sigma_1(t) :$$
$$sst_{s_i}^{[t_0;t_1]}(t) \cap cs(t) \neq \emptyset \Longleftrightarrow [\sigma_0(t) \leq sst_{s_i}^{[t_0;t_1]}(t) \leq \sigma_1(t)]$$

$rC2$: Runtime constraint $rC1$ is instantiated by the assumed user input:

$$\forall t : \sigma_0(t) = 25^\circ \wedge \sigma_1(t) = 40^\circ$$

$rC3$: The number of available satellites is assumed to be constant during execution. In fixed systems, this might be a system constraint but so it points out the approach's genericness:

$$\forall t : n = 3$$

$rC4$: The satellites' initial angles are assumed to be $\sigma_{s_0}^s = 70^\circ$, $\sigma_{s_1}^s = 190^\circ$ and $\sigma_{s_2}^s = 310^\circ$.

$rC5$: It is assumed the applications starts at $t_s = 0$.

$rC6$: The satellites' orbit is assumed to be 1000 km above the earth surface and their period of revolution is assumed to be 100 min. So, the resulting angular velocity of each satellite is $\omega_{s_i} = 0,06^\circ/s$ for $i = 0,1,2$. Together with runtime constraints 4 and 5, it instantiates the system constraint $sC2$.

Note that there are no time constraints given because the application is assumed to run continuously until it is stopped manually. So, the satellites STTs' time intervals are valid indefinitely long. It would be an easy task to introduce further runtime constraints.

4.3.3 Schedule

After specifying all constraints, the scheduling algorithm has to determine a feasible schedule that adheres to the given constraints by assigning explicit values to parameters that are not assigned so far. Here, only the observation systems' activations are not assigned. All other parameters are fixed according to system constraint $sC2$ and the runtime constraints. The resulting satellite schedule, that represents the node view, is:

$$s_0 : \begin{cases} obsystem_{s_0}^{on}(87\frac{1}{2}min + 100min \cdot k) \\ obsystem_{s_0}^{off}(91\frac{2}{3}min + 100min \cdot k) \\ send_{s_0}(91\frac{2}{3}min + 100min \cdot k) \end{cases}$$

$$s_1 : \begin{cases} obsystem_{s_1}^{on}(54\frac{1}{6}min + 100min \cdot k) \\ obsystem_{s_1}^{off}(58\frac{1}{3}min + 100min \cdot k) \\ send_{s_1}(58\frac{1}{3}min + 100min \cdot k) \end{cases}$$

$$s_2 : \begin{cases} obsystem_{s_2}^{on}(20\frac{5}{6}min + 100min \cdot k) \\ obsystem_{s_2}^{off}(25min + 100min \cdot k) \\ send_{s_2}(25min + 100min \cdot k) \end{cases}$$

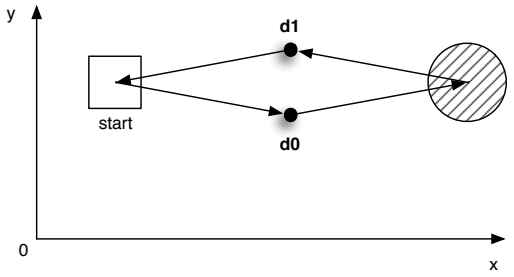

Fig. 3 Observation of the shaded area by two drones.

From this satellite schedule, the designated application's one, denoted by t_e^{app}, is derived. The user is only interested in this schedule because s/he is not aware of the explicit executing system in order to adhere to the system view. So, we gave the satellite schedule to point out the correlation between both different point of views.

$$t_e^{app} = \begin{cases} [20\frac{5}{6}min + 100min \cdot k; 25min + 100min \cdot k] \\ [54\frac{1}{6}min + 100min \cdot k; 58\frac{1}{3}min + 100min \cdot k] \\ [87\frac{1}{2}min + 100min \cdot k; 91\frac{2}{3}min + 100min \cdot k] \end{cases}$$

The number of revolutions is denoted by $k = 0,1,2,\ldots$. We omit the schedule's validation at this point because it obviously meets all constraints.

4.4 Drone Observation

4.4.1 Specific Assumptions

In the second example, several drones are intended to observe a certain area like the shaded one illustrated by Figure 3. This example differs from the satellite observation described in Section 4.3 because the drones' STTs are freely configurable so that there are more degrees of freedom for the scheduler. That is, the scheduling algorithm has to assign explicit values to more generic parameters than in the first example. We do not investigate how a constraint resolver might derive a feasible schedule. We point out that our programming approach allows to generically describe applications in order to abstract from the underlying executing system. The specific assumptions made to simplify matters in that example are:

1. The space-time is restricted to the three-dimensional case (two pace and one time dimension). Therefore, we denote the space-time trajectories $\overrightarrow{stt}_{obj_i}^{[t_0;t_1]}(t)$ in order to express the multiple space dimensions.
2. We abstract from the drones' detailed motions, i.e., takeoff and landing are not considered. Motions are free of acceleration and deceleration.
3. To simplify matters, the drones perform only rectilineal motions.

4.4.2 Constraints

In order to demonstrate the programming approach's genericness, we use the same application constraints introduced in section 4.3.2. Although the executing system is a different one, re-using them is allowed by programming against a virtualized executing system. However, the different executing system requires specifying other system constraints. Primarily, they generically describe the drones' motions as those are not fixed like the satellite ones' in Section 4.3.

*sC*1: There are n drones:

$$\exists n : R = \{d_0, d_1, \ldots, d_{n-1}\}$$

*sC*2: Each drone d_i's current position at time t is described by its STT $\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t)$ given an initial position $(x_0;y_0)^T$ at time t_0 and a velocity vector $\overrightarrow{v}_{d_i}^{[t_0;t_1]}$:

$$\forall d_i \in R; \forall t, t_0, t_1 :$$
$$\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t) = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \overrightarrow{v}_{d_i}^{[t_0;t_1]} \cdot (t - t_0) \quad (1)$$

*sC*3: The drones must not collide, i.e., their STTs must not be equal for a certain time. This only holds for drones' STTs' time intervals that overlap:

$$\forall d_i, d_j \in R; \forall t_0, t_1, t_2, t_3; \nexists t :$$
$$\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t) = \overrightarrow{stt}_{d_j}^{[t_2;t_3]}(t) \wedge ([t_0,t_1] \cap [t_2;t_3] \neq \emptyset) \wedge (i \neq j)$$

*sC*4: A drone's motion's velocity is limited to a maximum velocity $v_{d_i}^{max}$. So, two points in space must be reachable by moving with at most this velocity:

$$\forall d_i \in R; \forall t_0, t_1 :$$
$$\frac{|\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t_1) - \overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t_0)|}{|t_1 - t_0|} \leq v_{d_i}^{max} \quad (2)$$

The possibility to use equations (2) and (1) caused us to make assumption 3 in section 4.4.1.

*sC*5: This system constraint claims the motions' consistency. There must not be gaps in time or in space for two subsequent motions of the same object:

$$\forall d_i \in R; \forall t_0, t_1, t_2, t_3 :$$
$$\left((\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t_1) = \overrightarrow{stt}_{d_i}^{[t_2;t_3]}(t_2)\right) \wedge$$
$$([t_0;t_1] \cap [t_2;t_3] = \{t_1;t_2\}) \wedge (t_1 = t_2)$$

Analogously to the satellite example in section 4.3, runtime constraints instantiate the application.

*rC*1: A drone d_i is inside or outside $cs(t)$ at time t when its current position $\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t)$ is inside or outside an area within a certain radius r around a given space-time point $(x_m(t);y_m(t))^T$:

$$\forall d_i \in R; \forall t, t_0, t_1; \exists x_m(t), x_m(t) :$$
$$\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t) \cap cs(t) \neq \emptyset \iff$$
$$\left|\overrightarrow{stt}_{d_i}^{[t_0;t_1]}(t) - \begin{pmatrix} x_m(t) \\ y_m(t) \end{pmatrix}\right| \leq r \quad (3)$$

*rC*2: In order to complete runtime constraint *rC*1 we assume as user input:

$$\forall t : x_m(t) = 9 \wedge y_m(t) = 5$$

*rC*3: The number of drones remains constant:

$$\forall t : n = 2$$

*rC*4: The drones' initial positions $(x_0;y_0)^T$ at the application's starting time t_s are $(1;5)^T$ and $(0;5)^T$ for d_0 and d_1, respectively.

*rC*5: It is assumed the application starts at $t_s = 0$.

*rC*6: The drones' d_0, d_1 maximum velocities are assumed to be a general maximum velocity of four space units per time unit. This constraint counts to the runtime constraints because the maximum velocity might depend on (at programming time) non-predictable conditions like the weather:

$$\forall d_i \in R; \forall t_0, t_1 : |\overrightarrow{v}_{d_i}^{[t_0;t_1]}| \leq v_{d_i}^{max} = 4$$

4.4.3 Schedule

In contrast to the example in section 4.3, the drones' motions are not fixed. They are subject to the scheduling algorithm that has to assign explicit values to the generic motion description given in system constraint *sC*2. Their resulting schedule directly influences the application's one. One pos-

sible drone schedule (node view) is: ($t_c = \frac{\sqrt{17}}{4}$):

$$d_0 : \begin{cases} \overrightarrow{stt}_{d_0}^{[0;t_c]}(t) = (1;5)^T + \frac{1}{t_c} \cdot (4;-1)^T \cdot t \\ \overrightarrow{stt}_{d_0}^{[t_c;2t_c]}(t) = (5;4)^T + \frac{1}{t_c} \cdot (4;1)^T \cdot (t-t_c) \\ \overrightarrow{stt}_{d_0}^{[2t_c;3t_c]}(t) = (9;5)^T + \frac{1}{t_c} \cdot (-4;1)^T \cdot (t-2t_c) \\ \overrightarrow{stt}_{d_0}^{[3t_c;4t_c]}(t) = (5;6)^T + \frac{1}{t_c} \cdot (-4;-1)^T \cdot (t-3t_c) \\ \overrightarrow{stt}_{d_0}^{[4t_c;5t_c]}(t) = (1;5)^T + \frac{1}{t_c} \cdot (4;-1)^T \cdot (t-4t_c) \\ \overrightarrow{stt}_{d_0}^{[5t_c;6t_c]}(t) = (5;4)^T + \frac{1}{t_c} \cdot (4;1)^T \cdot (t-5t_c) \\ \overrightarrow{stt}_{d_0}^{[6t_c;7t_c]}(t) = (9;5)^T + \frac{1}{t_c} \cdot (4;1)^T \cdot (t-6t_c) \\ \ldots \\ obsystem_{d_0}^{on}(2t_c - \frac{1}{4}) \\ obsystem_{d_0}^{off}(2t_c + \frac{1}{4}) \\ send_{d_0}(2t_c + \frac{1}{4}) \\ obsystem_{d_0}^{on}(6t_c - \frac{1}{4}) \\ obsystem_{d_0}^{off}(6t_c + \frac{1}{4}) \\ send_{d_0}(6t_c + \frac{1}{4}) \\ \ldots \end{cases}$$

$$d_1 : \begin{cases} \overrightarrow{stt}_{d_1}^{[0;2t_c]}(t) = (0;5)^T + \frac{1}{2t_c} \cdot (1;0)^T \cdot t \\ \overrightarrow{stt}_{d_1}^{[2t_c;3t_c]}(t) = (1;5)^T + \frac{1}{t_c} \cdot (4;-1)^T \cdot (t-2t_c) \\ \overrightarrow{stt}_{d_1}^{[3t_c;4t_c]}(t) = (5;4)^T + \frac{1}{t_c} \cdot (4;1)^T \cdot (t-3t_c) \\ \overrightarrow{stt}_{d_1}^{[4t_c;5t_c]}(t) = (9;5)^T + \frac{1}{t_c} \cdot (-4;1)^T \cdot (t-4t_c) \\ \overrightarrow{stt}_{d_1}^{[5t_c;6t_c]}(t) = (5;6)^T + \frac{1}{t_c} \cdot (-4;-1)^T \cdot (t-5t_c) \\ \overrightarrow{stt}_{d_1}^{[6t_c;7t_c]}(t) = (1;5)^T + \frac{1}{t_c} \cdot (4;-1)^T \cdot (t-6t_c) \\ \ldots \\ obsystem_{d_1}^{on}(4t_c - \frac{1}{4}) \\ obsystem_{d_1}^{off}(4t_c + \frac{1}{4}) \\ send_{d_1}(4t_c + \frac{1}{4}) \\ \ldots \end{cases}$$

In this schedule, d_0 starts at $(1;5)^T$, moves to $(5;4)^T$, then to $(9;5)^T$, to $(5;6)^T$ and finally to $(1;5)^T$ again. d_1 starts at $(0;5)^T$. While d_0 is on the way, d_1 moves to $(1;5)^T$ and starts the same tour like d_0 just when d_0 leaves $(9;5)^T$, i.e., the reversal point of the tour. So, d_1 keeps a phase displacement of half tour time to d_0. The schedule could be continued arbitrarily by adding $(k \cdot \sqrt{17})$ to each given time except d_1's very first STT. Landing both drones would require to schedule d_1's motion so that it lands at $(0;5)^T$. From that drones' schedule, the application's execution time t_e^{app}'s schedule (system view) is derived:

$$t_e^{app} = \left[\frac{\sqrt{17}}{2} - \frac{1}{4} + \frac{\sqrt{17}}{2} \cdot k; \frac{\sqrt{17}}{2} + \frac{1}{4} + \frac{\sqrt{17}}{2} \cdot k \right]$$

Here $k = 0, 1, 2, \ldots$ denotes the number of half tours both drones complete, i.e., k increases two times per drone's tour. Again, we abstain from the schedule's detailed validation and ensure that it meets all constraints.

Recapitulatory, the same application description and constraints are used to describe the observation of a spatial area. This is independent of the executing system and especially from its distribution. Instead, the programmer and user refer to a virtualized executing system according to the system view. It is not necessary to give constraints that describe the sub-systems' cooperation or application migration explicitly. The system constraints describe the sub-systems' motions generically while the runtime constraints instantiate these constraints as far as possible. The overall instantiation is done by the scheduling algorithm.

5 Related Work

Already existing operating systems for distributed systems like *Amoeba* [14], *Plan 9* [10] and *Emerald* [6] follow the approach of migration and distribution transparency. Similar to our approach, a programmer doesn't need not to be aware of different locations within the system. Unfortunately, these systems aim at stationary executing systems, i.e., they are not intended to execute motion aware applications so that applications' motion awareness is lacking in these systems.

At first glance, our project and swarm approaches like the operating system *SwarmOS* [7] want to achieve common purposes. FlockOS differs from these projects as we want to realize a top-down approach whereas the others follow a bottom-up approach. There, several pre-defined behavior patterns are assigned to the sub-system the executing system comprises. So, a programmer is aware of the system's distribution, i.e., s/he has a node view in mind. SwarmOS also suffers from the lack of the applications' motion awareness. We want to take an important step forward by providing the possibility to hide this distribution and to consider the applications' motions within our operating system explicitly.

A fundamental approach to formalize spatiotemporal constraints can be found in Allen [1]. It introduces an interval-based temporal logic that allows reasoning about time intervals on basis of 13 basis relationships between these intervals. In a similar way, spatial information are handled by the region connection calculus (RCC) [11]. [4] investigates spatiotemporal reasoning on basis of RCC and Allen's interval algebra. Work on (qualitative) modeling of moving objects can be found in geographical information systems [15] and data base systems [12], inter alia. Further research has to investigate imposing quantitative motion representation on these approaches.

As the spatiotemporal constraints introduced in this paper are intended to extend common (imperative) general purpose programming languages, further investigation concerns developing a suitable constraint formalism and integrating it into an imperative programming language, e.g., C. Promising approaches can be found in [2] and [13]. These papers introduce the so-called *Constraint Handling Rules*. One typical application domain of these Constraint Handling Rules is spatiotemporal reasoning so that this would be a good

starting point for our further research. The multi-paradigm Mozart Programming System and its language Oz provides another promising possibility. Existing work on constraint-based scheduling like [9] and [5] has to be investigated in order to develop or to adapt an existing constraint resolver so that our spatiotemporal constraints can be handled.

6 Conclusion

Within this paper, we introduced spatiotemporal constraints as a suitable programming abstraction for distributed mobile systems. They allow describing applications in real space-time while preserving the distribution transparency within the executing system and providing the necessary motion awareness within the application. We pointed out that such a top-down programming approach allows realizing generic application descriptions by adhering to the introduced system view. The distinction between three different constraint types underlines the claimed genericness that is demonstrated by two examples as well as the programming model's close relationship with scheduling in real space-time.

The next steps beyond the scope of this paper include investigation of possibilities to formalize the spatiotemporal constraints with regard to correctness reasoning and integration into general purpose programming languages. Other points of research concern appropriate scheduling algorithms and in particular the approach of staged constraint resolving, respectively. With regard to robustness and self-stabilization, an appropriate scheduling algorithm to be developed should work decentralized and online. Implementing FlockOS as an operating system that supports high portability and efficiency, especially in terms of real space-time scheduling, is aspired by our research group.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
2. T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.
3. T. Frühwirth and S. Abdennadher. *Constraint Programmierung – Grundlagen und Anwendungen*. Springer, Berlin/Heidelberg, 1997.
4. A. Gerevini and B. Nebel. Qualitative spatio-temporal reasoning with rcc-8 and allen's interval calculus: Computational complexity. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, 2002.
5. H. H. Hesselink and S. Paul. Planning aircraft movements on airports with constraint satisfaction, 1998.
6. E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Transactions on Computer Systems*, 6(1):109–133, 1988.
7. J. D. McLurkin. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2004.
8. ODP. Basic Reference Model of Open Distributed Processing. Part 3: Architecture, ITU-TX. 903-ISO/IEC 10746-3: ISO/IEC JTC1/SC2, 1995.
9. C. L. Pape and I. S. A. Constraint-based scheduling: A tutorial.
10. R. Pike, D. Presotto, K. Thompson, and H. Trickey. Plan 9 from bell labs. *Computing Systems*, 8(3):221–254, 1995.
11. D. A. Randell, Z. Cui, and A. Cohn. A Spatial Logic Based on Regions and Connection. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 165–176. Morgan Kaufmann, San Mateo, California, 1992.
12. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. pages 422–432, 1997.
13. J. Sneyers, P. Van Weert, T. Schrijvers, and L. De Koninck. As time goes by: Constraint Handling Rules – A survey of CHR research between 1998 and 2007. Submitted to *Journal of Theory and Practice of Logic Programming*, 2008.
14. A. Tanenbaum, R. v. Renesse, H. v. Staveren, G. Sharp, S. Mullender, A. Jansen, and G. v. Rossum. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 33:46–63, 1990.
15. N. Van, D. Weghe, A. G. Cohn, G. De, T. É, and P. D. Maeyer. A qualitative trajectory calculus as a basis for representing moving objects in geographical information systems. *Control and Cybernetics, 35(1):97–119*, 2006.
16. A. Wegener, E. M. Schiller, H. Hellbrück, S. P. Fekete, and S. Fischer. Hovering data clouds: A decentralized and self-organizing information system. In H. De Meer and J. P. G. Sterbenz, editors, *Proceedings of the 1st International Workshop on Self-Organizing Systems (IWSOS 2006)*, volume 4124 of *Lecture Notes in Computer Science*, pages 243–247, Berlin/Heidelberg, Germany, Sept. 2006. Springer.
17. M. Werner, D. Müller, M. Däumler, J. Richling, and G. Mühl. Operating system support for distributed applications in real space-time. In R. Chbeir, Y. Badr, A. Abraham, D. Laurent, and F. Ferri, editors, *Workshop on Autonomous and Autonomic Software-based Systems (ASBS) at the IEEE International Conference on Soft Computing as Transdisciplinary Science and Technology*, pages 469–478, 2008.
18. M. Werner, J. Richling, D. Müller, and G. Mühl. Towards a holistic approach for distributed real space-time systems. In *International Workshop on Dependable Network Computing and Mobile Systems at the 27th International Symposium on Reliable Distributed Systems*, pages 63–67, 2008.