The Contextual Map - A Context Model for Detecting Affinity between Contexts

Robert Schmohl and Uwe Baumgarten

Institut für Informatik, Technische Universität München
Garching bei München, Germany
schmohl@in.tum.de, baumgaru@in.tum.de

Abstract. Context-awareness represents an important research domain in mobile computing by utilizing information about persons, places and objects anytime and anywhere. The highly dynamic contexts created by this paradigm raise questions how to efficiently determine alikeness and affinity between such contexts. Inspired by mechanisms from location-aware computing, we tackle the issue of contextual proximity by constructing an n-dimensional map-model, which serves as a context model for regular context repositories. This Contextual Map enables us to store non-location contexts in a map-based way. Further, this model enables us to conduct location-based n-dimensional proximity detection on the non-location contexts, hence giving us the possibility to determine contextual proximity. This paper introduces the contextual map model, describing how principles from the location-based service domain can be leveraged on general context-aware computing and how they can be employed to detect affinities between different contexts.

Keywords: Context-aware computing, contextual affinities, contextual boundaries, proximity detection.

1 Introduction

Context-aware computing focusses on utilizing any information describing the current *context* of an entity, which may be a place, a person, an object, etc. In practice, this information comprises of the situation and possible actions of the entity, derived from its current surrounding, which is captured by sensors of devices associated with the entity, or the infrastructure hosting such devices [1,3]. To store and exploit such contextual information, context-aware systems utilize sophisticated context models. Those models represent the context captured from the real-world in a way suitable for further processing, such as identifying contextual coherences and inferring new context.

Context-awareness is applicable in numerous application domains. Especially the utilization of mobile devices emphasizes the exploitation of the highly dynamic environment in which mobile hosts usually roam. However, providing information about the most current context to mobile applications is an elaborate effort. For this reason, context-aware systems are usually middleware solutions capable of acquiring and managing contextual data autonomously, hence providing the resultant context to high-level applications via a dedicated interface.

In order to identify similarities among individual entities' contexts we employ techniques known from location-based computing. Although exhibiting a high degree of specialization, map-based location models yield principles, which can be applied to conventional context-aware computing as well. We have discovered that the principle of *proximity* in the location domain can be applied to face certain aspects of context-awareness, too. The geographical proximity in the location domain expresses that entities are close to each other. By projecting this setting on more general contexts, *contextual proximity* may express that contexts are alike or affine, hence "close" to each other.

This approach has yielded the conceptualization of the contextual map model, abstracting contextual information into an n-dimensional map, thus assigning each piece of context a *position* in this map. Since this context model represents contextual information in a n-dimensional cartesian map model, it allows the application of mechanisms known from location-aware computing on non-location contextual information.

For the identification of contextual proximity, we calculate the Euclidean distance between n-dimensional contexts to determine their degree of alikeness, such as we would calculate the distance between locations of objects in 3-dimensional space. In our approach, we define *context boundaries* as the degree of alikeness between different contexts. More precisely, we monitor dynamic affinities between pieces of context according to their changing *distances* to each other in the contextual map. Those distances allow us to determine, whether context boundaries have been crossed, hence whether contexts become more affine (converging) or less affine (separating). Figure 1 sketches the idea behind the contextual map.

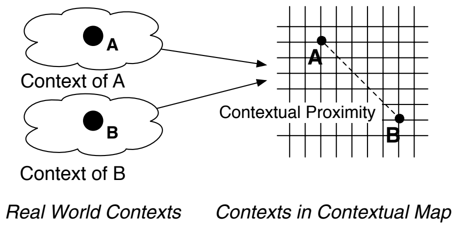

Fig. 1. Contexts in the Contextual Map

This paper explores the application possibilities of the contextual map model as a middleware solution on mobile hosts. Section 2 enumerates the work related to our project. Section 3 introduces the principles from the domains of location- and context-awareness, that we base our work on. With this information given, we introduce our contextual map model in section 4. Section 5 explains, how the contextual map is employed to monitor affinity between contexts and how this is related to contextual boundaries. Subsequently, section 6 summarizes the overall workflow of employing the contextual map model before the paper is concluded in section 7, giving an outlook on the enhancements of the approaches presented here.

2 Related Work

Prior to the conceptualization of the work presented here, we have surveyed the domain of context-aware computing deriving an architecture that allows a generalized view on the domain [10]. Here, we are presenting a novel concept of a context model exploiting principles of the location-aware computing domain. Since it is quite specialized in nature, it is to be regarded supplemental to well proven context models [12], such as ontologies [1,2,4,5].

Proximity and separation detection allows the definition and utilization of context boundaries when employed with the contextual map model. Such context boundaries define *distances* between contexts in the contextual map hence specifying *degrees of affinity* between those contexts. Roman et. al [9] have conducted similar work by defining thresholds where exceeding of those thresholds equals crossing of such boundaries. Proximity and separation detection, as presented by Küpper and Treu [7], requires frequent location updates of the involved mobile hosts. Besides the common update semantics including polling, periodic updates, distance- and zone-based updates, there is a common endeavor on designing update semantics, which minimize the amount of updates while keeping the location of the target as current as possible [7,6]. Küpper and Treu present efficient ways to achieve this objective by employing sophisticated algorithms with circular and strip-like update zones in 2-dimensional space.

A notable alternative to our work is depicted by the theory of context spaces introduced by Padovitz et al. [8]. As the contextual map, context spaces aim at the multi-dimensional representation of contextual attributes. Multi-dimensional context spaces are partitioned into regions denoting bounds of specific contextual situations. Concrete context states mapped from real world entities are then associated to such context regions according to their proximity in the context space.

3 Background

3.1 Context-Aware Computing

Context-aware systems usually build upon the following fundamental workflow visualized in figure 2. They capture information about their surrounding from *context sources*, such as sensors, the network infrastructure, auxiliary software, etc. Subsequently, this information is refined by relating this information to the context in question. This task is usually performed by a dedicated component, namely the *context capturing interface*. The resultant *contextual information* is stored in *context models*. *Inference engines* use this data to derive new contextual information according to application-specific rules. The data derived at the

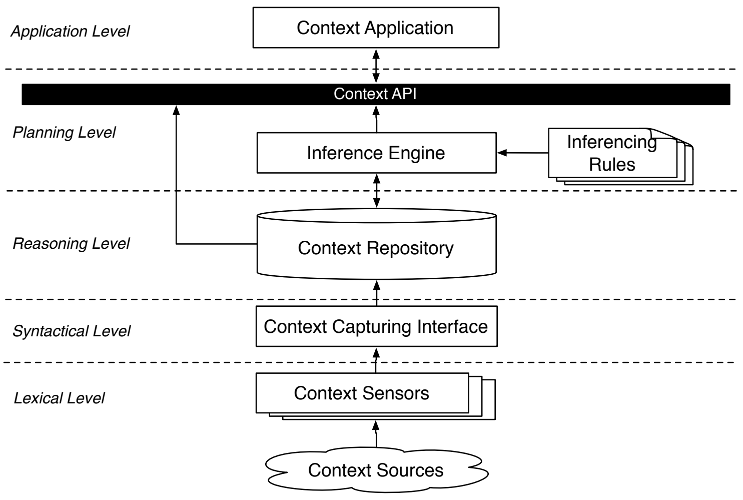

Fig. 2. Architectural Draft for context-aware computing

end of this process is made available by *context APIs* to any application utilizing the most current context. As stated, figure 2 shows a generic architecture utilizing this workflow [10]. In summary, the context API provides transparent access to a context-aware middleware, which utilizes all contextual data from the attached context sources autonomously. This approach facilitates the development of context-aware application and enables portability of a context-aware system.

3.2 Proximity and Separation Detection in 2-Space

The detection of mobile entities approximating or departing each other depicts important application cases for location-based services [7]. As with all of the services implying location, proximity and separation detection requires frequent location updates of the involved mobile hosts. Based upon the location committed by the targets, proximity or separation can be detected. There are four basic mechanisms on how to trigger a location update [7]: polling, periodic updates, zone-based updates and distance-based updates.

A common endeavor on designing update semantics is the minimization of the amount of updates while keeping the location of the target as current as possible [7,6]. There are two efficient ways to achieve this objective in combination with detecting proximity or separation of mobile entities [7]. Both conduct the detection mechanism following a mobile entity's location update, which work as followed. The first approach is based on circular areas centered around mobile hosts. Location updates are committed upon leaving those areas. Proximity and separation are detected upon calculating the smallest (proximity) and largest (separation) possible distances, which both depend on the circles' radii. The second approach focuses on defining an orthogonal strip between two mobile entities

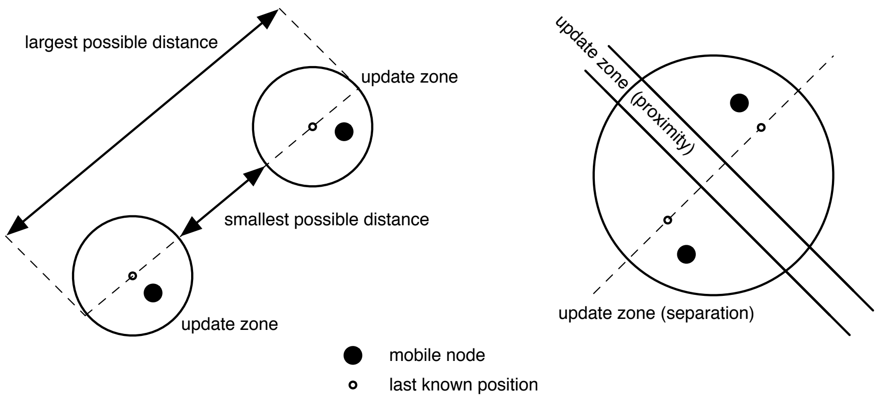

Fig. 3. Proximity and separation detection

and spanning a circle around the center of the line between them. Proximity is detected upon a location update triggered by one of the targets entering the strip. Separation detection is conducted after a location update committed by a target leaving the circular zone. Figure 3 illustrates both approaches.

4 The Contextual Map Model

In this section we introduce the contextual map model for storing contextual information in a context repository, as depicted in figure 2. After presenting the model's structural characteristics we focus on how contextual information is mapped into this model. To improve the reader's understanding, we are going to illustrate the working principle of the contextual map by employing an example, which exploits the context of a weather station.

4.1 Composition

The key idea of the contextual map model is to represent the context of entities in an n-dimensional realm. The context of a single entity corresponds to a single entry in the contextual map. Such an entry is composed of n coordinates and hence can be interpreted as a *position* in the map. Referring to our example, the context of our weather station denotes the current weather conditions at a specific location. Its context represents a single position in the contextual map, whereas the context of another weather station denotes another map position.

An entity's context, captured by various context sensors, is mapped into the n dimensions of the contextual map by employing a particular *mapping function*. We are going to sketch such a function in the subsequent section 4.2. After mapping the information into the contextual map, the entity's current context is represented by n coordinates inside the map. Analogously, one can imagine a 3-dimensional map of space, which is employed to represent the *location* of entities in space. We extend this 3-dimensional model by adding additional dimensions, in order to include further contextual information other than location. Finally, each dimension corresponds to an elementary contextual attribute of a complete piece of context. In the contextual map, a single dimension is the smallest unit of context representation.

To further augment our model, we allow the contextual map to be partitioned into *ranges.* A range groups the map's dimensions, which share typological similarities. For instance, our weather station's location is a piece of context, decomposing into three subordinate building blocks corresponding to the location's cartesian coordinates. In the contextual map, three dimensions can be dedicated for coordinate representation, grouping them to one range dedicated to the representation of locations. Since the weather station's context includes more than merely its location, additional ranges must be included. E.g., an additional range may include the station's environmental readings with a single dimension being dedicated to each sensor (e.g. temperature, humidity, etc.). Table 1 sketches this example.

The dimensions in the contextual map may be regarded as its axes, requiring well defined units of representation. The choice of those is virtually unrestricted as long as it fits the contextual attribute, which it is supposed to represent. However, within ranges units have to be uniform spawning a d-dimensional cartesian coordinate system for each range with d being the amount of the range's dimensions. E.g., the dimensions of the weather station's location range all need to be defined in kilometers, whereas all of the environment range's dimensions have to be defined in a uniform way representing the particular weather condition (see table 1). This is necessary to enable efficient detection of contextual affinity as discussed later in section 5.

The n-dimensional representation of an entity's context in the map is atomic and thus exploitable for further analysis. Changes of its context can alter the contextual correlation to other entities in regard to contextual boundaries. Since contextual boundaries usually define the scope of *similar context*, we can employ mechanisms known from 2- or 3-dimensional map models to detect proximity among contexts inside the contextual map, hence identifying contextual affinities by identifying *proximate context*, as explained later on in section 5.

4.2 Context Mapping

Based on the context model introduced in the previous section 4.1, it remains to be clarified how contextual information is mapped into the contextual map. The basic approach is to employ a *mapping function*, which inputs quantified contextual data from according context sources and maps them to cartesian coordinates of the diverse ranges in the contextual map. Formally, given an entity, its context is mapped to the contextual map as followed:

$$m : s \to \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix} | s \in S, \boldsymbol{v} \in V_n$$

Table 1. Example ranges

Range	# of dimensions	Axis definition
Location	3 (x, y, z)	Cartesian coordinates (km)
Environmental data	4 (temperature, barometric pressure, humidity, wind speed)	0 (min) to 100 (max)

where S is the set of quantified context source data and V_n the correspondent n-dimensional realm in the contextual map. $\boldsymbol{v}$ represents a single context in the contextual map with $\boldsymbol{v}$ corresponding to the entity's real-world context. According to the fragmentation of the contextual map into ranges discussed in the previous section 4.1, m needs to map individual contextual data to the correspondent ranges in V_n.

It is to be noted, that S may cover a very heterogeneous spectrum of contextual sources [11] implying heterogeneous definitions of m as well. In the rest of the paper we proceed with a mapping technique tailored to our weather station example. We assume that the context of an entity is typified and hence given by a set of scalars, each corresponding to its according type of context.

Given our weather station example, we first define a representation for its context in the contextual map model and define a mapping function for the contextual data to be mapped into that map, subsequently. We proceed as followed. First, we define two context types: location and environmental data. We assign each type a range with an according amount of dimensions in the contextual map, as shown in table 1. Consequently, we assign the dimensions of each range uniform axis definitions.

With this setting, it is possible to define a mapping function m, which transfers the weather station's context into the contextual map. First of all, we require m to handle each context type and its corresponding range separately. m takes the quantified contextual data captured and derived from the weather station's context and calculates its position in the contextual map adhering both range and axis definitions. Mapping location into the map is quite straight forward: The station's coordinates are translated to cartesian coordinates in the map. The coordinates for the environmental data are calculated by determining the relation of the current value to the pre-defined extremes. Let temperature to be allowed to adopt a value between -50°C and 50°C resulting in 100 adoptable units. With the axis of dimensions of the environment range ranging from 0 to 100, a currently measured temperature of 25°C corresponds to a coordinate of 75 for the temperature dimension. The mapping of the remaining environmental measurements works analogously.

The mapping function presented here covers a special use case, and must not be seen as universally valid. The heterogeneous spectrum of context-aware application cases [10,11] requires m to be adapted individually depending on the required application cases. A large part of this heterogeneity comes from the heterogeneous environment, which usually surrounds context-aware systems. Given the assumption, that contextual data is captured and quantified, m is to be defined in a way to specifically handle the input given by that quantification.

Since the context of entities is dynamic, the mapping must be employed iteratively to ensure the most current context to be stored in the contextual map. Such context actuality is achieved by frequent contextual updates. The according update semantics are discussed later on in section 5.3.

5 Application of the Contextual Map

So far, we have solely depicted the contextual map model in the previous section 4. In this section we discuss the applicability of our context model. We assume that entities possessing their own contexts utilize mobile devices, which acquire and manage such contexts and commit contextual updates to distribute this information. In the following we first discuss the definition of context boundaries, which form the basis for enabling triggers based on contextual proximity/affinity. With context boundaries defined, the principles of measuring the proximity of contexts are introduced, hence allowing to determine the affinity between those contexts. In addition, we reflect on efficient update semantics, to derive the current context to be most current despite of minimized efforts.

5.1 Context Boundaries

Contextual boundaries represent the degree of alikeness between contextual information. More precisely, contextual boundaries may be defined between different entities to determine the affinity of their individual contexts. This affinity expresses how interesting or - contrarily - uninteresting another entity's context is. Concluding, context getting *closer* also raises its relevancy for the concerned entity [9].

For example, a car driver may be interested in proper weather conditions on his route, hence taking action if a proximate weather station reports strong weather. We define the contextual boundaries between the entities *car driver* and *weather station* on two ranges: location (R_{loc}) and environment (R_{env}) (compare table 1). Both ranges compose the context of each entity. An entitiy's physical

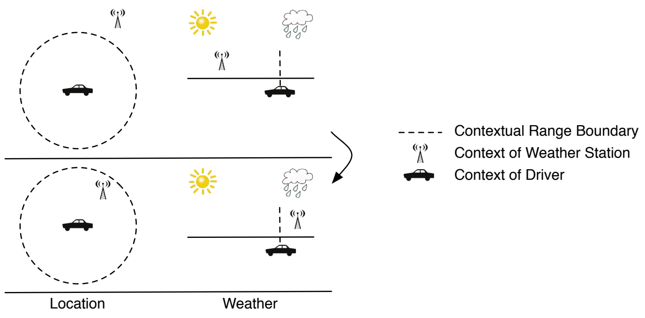

Fig. 4. Crossing Context Boundary Example

position is represented in R_{loc}. R_{env} represents a weather station's currently measured environmental readings and the driver's personal conception about not-agreeable weather. Now, we can define a context boundary, which delimits the driver's willingness to drive according to how bad the weather is. The boundary concerning R_{loc} expresses a weather station's distance to the driver, so that it actually becomes of his interest. The boundary on R_{env} focuses on how close the current weather at a station may come to the weather conditions regarded as critical by the driver. The contextual boundary between the entities *driver* and *weather station* gets crossed if the driver gets close to the weather station, which reports conditions close to the driver's critical criteria. Figure 4 illustrates the definition and crossing of this example boundary.

Given this example, we can formally define a contextual boundary B:

- A *proximity threshold* t_i is defined on each contextual range R_i, which is relevant to B. Each t_i is a single value dependent on all of the R_i's dimensions, hence, defining the degree of alikeness of R_i's contexts. Basically speaking, the threshold represents the *distance*, which delimits the range's contextual information to be "proximate" or not.
- The set of thresholds t_i of all ranges $R_1, ..., R_i, ..., R_m$ relevant to B defines the contextual boundary: $B = (t_1, ..., t_m)$

5.2 Detecting Affinity of Contexts

With the principle of contextual boundaries defined, we proceed with describing how contextual boundaries can be exploited using the contextual map model. As we have argued earlier, crossing context boundaries corresponds to different entities' contexts getting either more ore less affine (or rather contextually "closer/farther" to/from each other), thus crossing the affinity degree specified by the boundary.

With context boundaries defined in a contextual map, the next step is to determine crossings of those boundaries indicating changes of contextual affinities. Each of a context boundary's thresholds is defined on a single range. For this reason the check for two contexts having crossed the boundary by converging or separating from each other is performed range-specifically. Given two entities P and Q, their respective contexts P and Q are partitioned into multiple ranges. Since ranges group contextual data of equal types, we compute the *distances* between the two contexts inside each range separately (only ranges affected by the boundary). Such a range-level distance between P ands Q is represented by the Euclidean distance D:

$$D = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{1}$$

with p_i and q_i being the coordinates of dimension i of the contexts P and Q, respectively, and d denoting the number of dimensions of the concerning range.

With the distance D calculated, we employ proximity and separation detection [7] to determine, if the contexts of the two entities are "closing" or "separating" on range level. The thresholds of a contextual boundary serve as limits to enable proximity or separation alerts. Such an alert is triggered when all of the boundary's thresholds have been breached, i.e. when the boundary has been crossed on each relevant range. The "direction" of exceeding a context boundary's thresholds declares whether P and Q have *converged* or *separated* while crossing the context boundary, hence getting either more or less affine, respectively.

Returning to our weather station example, we assume, that the contexts of both the car driver and surrounding weather stations are iteratively captured and mapped to the contextual map. Hence, each weather station and the car driver get a constantly updated contextual map position associated with their respective context. The car driver has defined a contextual boundary defining dangerous weather, at which he is not willing to drive (i.e. weather coming close to his critical weather definition). Hence this boundary is decomposed into two thresholds defined on two ranges as depicted earlier in section 5.1 and figure 4: on R_{loc} expressing the driver's tolerated distance to a weather station reporting bad weather, and on R_{env} defining the tolerance interval to weather conditions at which the driver refuses driving. Let the boundary's thresholds be further defined as 10 kilometers on R_{loc} and 15 units on R_{env} (as depicted in table 1). The distances D between the contexts of driver and weather station are computed for each range upon receiving a contextual update. In this example, such an update may yield changing weather conditions and/or changing driver's location. As soon as the driver gets closer than 10 kilometers to a weather station, which reports weather less than 15 units distant to the driver's critical weather definition, the defined context boundary is crossed and a proximity alert is triggered. Hence, the driver may take action changing his route or stop driving.

5.3 Update Semantics

An obvious precondition for detecting proximity/separation of contexts in the contextual map is the knowledge about its most current positions in the map. Context actuality is achieved by contextual updates, i.e. updates containing an entity's most current context information. The challenge is to define efficient update semantics, so that a minimal amount of updates delivers the most current context possible. E.g., frequent updates issued in short intervals deliver very actual context information, but are highly inefficient, since most of them are redundant due to missing context changes.

In section 3.2, we have outlined efficient semantics for issuing location updates to detect geographical proximity among mobile hosts [7]. Those mechanisms allow the acquisition of the most current context with a minimal amount of location updates. With little effort, we can transfer those principles on the contextual map. Both the geographical setting in section 3.2 and the contextual map are settled in multi-dimensional Euclidean space (with the geographical setting being 2-dimensional expressing width and depth). The proximity and separation detection mechanisms from section 3.2 work zone-based based on circles and strips. Extending those mechanisms on the contextual map concludes the definition of those on d-dimensional hyperspheres and hyperplanes:

- *Hyperspheres method:* In the 2-dimensional realm ($\mathbb{R}^2$), contextual updates are sent upon leaving the circular zone spawned around the according entity's position during the last update (last known position). Extending this principle on the d-dimensional realm ($\mathbb{R}^d$), we define a d-dimensional hypersphere around this position in the contextual map. This position forms the center point C of a hypersphere with its radius r denoting the update zone. This definition allows us to determine, if the current context manifests itself at a position outside the hypersphere, i.e. if the Euclidean distance between the current context P and C is greater r. Formally, a contextual update is triggered upon the following condition:

$$\sqrt{\sum_{i=1}^{d}(P_i - C_i)^2} > r$$

- *Hyperplanes method:* For proximity detection in the geographical realm, strips are spawned orthogonally between two mobile nodes eligible for proximity detection. Contextual updates are sent by a mobile node upon entering such a strip. Thus, the update zone is bounded by two lines. The d-dimensional analogy requires the definition of two $(d-1)$-dimensional hyperplanes, bounding the update-zone in $\mathbb{R}^d$. Since this approach is still settled in Euclidean space, we can formally define those two hyperplanes in $\mathbb{R}^d$ analogously as in $\mathbb{R}^3$. Let there be two contexts, P and Q, with according coordinates in the contextual map. First, the middle point C between those contexts is to be defined. Now, the hyperplanes need to be defined orthogonally to the line connecting P and Q, and they need to be equidistant from the the center point C, in order to position the strip exactly in between P and Q. With the update zone set we can now trigger a contextual update, if one of both contexts enters the strip-area bounded be the two defined hyperplanes. However, this method is only feasible to conduct proximity detection between the two contexts. For separation detection the strip-method cannot be applied, as argued in section 3.2. In order to conduct separation detection, we define an update zone bounded by a hypersphere, centered at C. The contextual update is then triggered upon a context "leaving" the hypersphere, as we have discussed this process earlier already.

To demonstrate the applicability of the update semantics presented here, we return to the example with the driver and the weather station. We have two contexts, the driver D and the station S. We further regard the two relevant ranges in the contextual map: weather conditions R_{env} and location R_{loc}. However, the dynamic changes, which take place here are restricted to the weather station's current weather and the driver's location. Concluding, we have to pay attention to D's position in R_{loc} and S's coordinates in R_{env}, only. To proceed with this example we employ an update zone bounded by hyperspheres. Therefore, hyperspheres are defined around D and S on their respective ranges R_{loc} andR_{env}. For D, it seems reasonable to select a radius of 10 on R_{loc}, denoting an update necessity from the driver every 10 kilometers (neglecting the third dimension z as denoted in table 1 for reasons of simplicity). For the update-sphere of S, we define a radius of 5 on R_{env}. This corresponds to an update triggered when the weather conditions at the weather station change by $\frac{1}{20}$ of the scale (see range's axis definition in table 1).

6 Overall Workflow

With the basic working principle of the contextual map model discussed, we are now about to summarize the general workflow of detecting affinities between contexts.

1. *Context capturing*: The first step consists of abstracting the environmental context into the context model. The contextual map has to be set up according to this context, i.e. ranges, its axes are to be defined (section 4.1). Subsequently, the initial context is mapped into the contextual map. This includes the identification of entities possessing their own definable contexts and mapping them into the contextual map (section 4.2). As a result, every entity possesses a *position* in the map, corresponding to its initial context.
2. *Setting of contextual boundary:* In the next step, the contextual boundaries have to be defined, so that contextual affinity detection becomes possible. This includes the identification of ranges affecting the according boundary and defining the delimiter thresholds on the actual ranges (section 5.1).
3. *Definition of update semantics:* The entities' terms for committing updates about their most current contexts must be defined. This regards the selection of the appropriate bounding model for the update zones in $\mathbb{R}^d$, (hypersphere and hyperplane models in section 5), as well as the size of the update zones (hypersphere = diameter, hyperplanes = distance between each other).
4. *Monitoring current context:* It is assumed, that a mobile host is attached to each entity. It keeps track of the entity's current context, hence maintaining a contextual map representation of its entity locally. The mobile host captures the most current context from its context sensors, quantifies it in the context capturing interface (see figure 2), and merges the resultant data into its local contextual map (section 4.1). The entity's changing context corresponds to changing coordinates of its contextual position in the map.
5. *Determine contextual update:* The constant local monitoring of the entity's context enables the determination to commit a contextual update. This is exactly the case, when the entity's context in the local contextual map leaves the update zone (section 5). A contextual update is committed to the system by the entity's mobile host.
6. *Proximity/separation detection:* The distances from the entity's context to other contexts are determined by calculating the Euclidean distance (5.2). This process is conducted on each range affecting a context boundary.

7. *Trigger proximity and separation alerts:* The calculated distances are checked with the affected contextual boundaries. If a change of distance equals a crossing of a boundary (section 5.1), a proximity or separation alert is triggered. The alert is available for applications utilizing the context API as depicted in figure 2.

Figure 5 illustrates the workflow described above as a UML activity diagram.

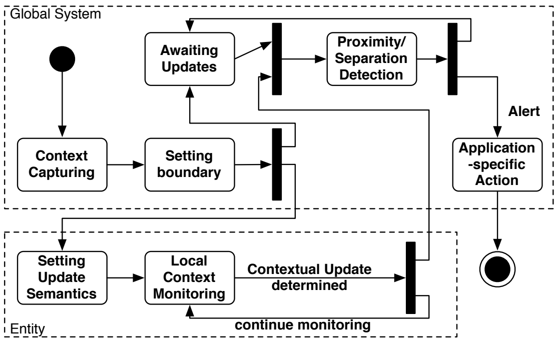

Fig. 5. Workflow

7 Conclusion and Outlook

With the contextual map, we have proposed a novel context model concept facilitating the work with contextual affinity. We have proposed a definition for context boundaries consisting of the degree of alikeness between different contexts based on typified ranges of contextual information. Together with the contextual map model, we are able to track specific alikeness of contexts and put it into relation with contextual boundaries. This process finally enables us to enrich the context API for context-aware applications (see figure 2) by providing contextual triggers, which report that a specified degree of contextual alikeness has been reached. The contextual map suggest the implemented as middleware concept augmenting existing context model by providing contextual affinity management.

With the conceptual sketch available, the next step encompasses the refinement of selected aspects of the conceptual map, especially the following two: First, a major issue concerns the heterogeneity of the mapping function. Handling heterogeneous context sources allows a more standardized process of mapping context to the map. The second issue in question regards the aspect of distribution. Although clearly being applicable to pervasive computing environments, we have not yet proposed architectures on how to distribute the contextual map model and its peripheral components on mobile nodes, networks infrastructures and so on.

Our most significant mid-term goal is the construction of a prototype model, which may allow a proof-of-concept simulation. This goal will then allow us to analyze issues on how to integrate the context model in proven context-aware systems.

References

1. Christopoulou, E., Goumopoulos, C., Kameas, A.: An ontology-based context management and reasoning process for ubicomp applications. In: sOc-EUSAI 2005: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence [2], pp. 265–270 (2005)
2. Christopoulou, E., Kameas, A.: Gas ontology: An ontology for collaboration among ubiquitous computing devices. Journal of Human-Computer Studies [1], 664–685 (2005)
3. Gellersen, H.W., Schmidt, A., Beigl, M.: Multi-sensor context-awareness in mobile devices and smart artifacts. Mob. Netw. Appl. 7(5), 341–351 (2002)
4. Rezwanul Huq, M., Thanh Tuyen, N.T., Lee, Y.-K., Jeong, B.-S., Lee, S.: Modeling an ontology for managing contexts in smart meeting space. In: SWWS 2007: Proceedings of the 2007 International Conference on Semantic Web and Web Services (2007)
5. Khouja, M., Juiz, C., Lera, I., Puigjaner, R., Kamoun, F.: An ontology-based model for a context-aware service oriented architecture. In: SERP 2007: Proceedings of the 2007 International Conference on Software Engineering Research and Practice (2007)
6. Kieß, W., Füßler, H., Widmer, J., Mauve, M.: Hierarchical location service for mobile ad-hoc networks. SIGMOBILE Mob. Comput. Commun. Rev. 8(4), 47–58 (2004)
7. Küpper, A., Treu, G.: Efficient proximity and separation detection among mobile targets for supporting location-based community services. SIGMOBILE Mob. Comput. Commun. Rev. 10(3), 1–12 (2006)
8. Padovitz, A., Loke, S.W., Zaslavsky, A.: Towards a theory of context spaces. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004, vol. 1, pp. 38–42 (March 2004)
9. Roman, G.-C., Julien, C., Huang, Q.: Network abstractions for context-aware mobile computing. In: ICSE 2002: Proceedings of the 24th International Conference on Software Engineering, pp. 363–373. ACM Press, New York (2002)
10. Schmohl, R., Baumgarten, U.: Context-aware computing: a survey preparing a generalized approach. In: IMECS 2008: Proceedings of the International Multi-Conference of Engineers and Computer Scientists 2008, International Association of Engineers (2008)
11. Schmohl, R., Baumgarten, U.: A generalized context-aware architecture in heterogeneous mobile computing environments. In: The Fourth International Conference on Wireless and Mobile Communications, 2008. ICWMC 2008, vol. 1, pp. 118–124 (August 2008)
12. Strang, T., Linnhoff-Popien, C.: A context modeling survey. In: Davies, N., Mynatt, E.D., Siio, I. (eds.) UbiComp 2004. LNCS, vol. 3205. Springer, Heidelberg (2004)