Modeling Service-oriented Context Processing in Dynamic Body Area Networks

Clemens Lombriser, *Student Member, IEEE,* Raluca Marin-Perianu,
Daniel Roggen, *Member, IEEE,* Paul Havinga,
Gerhard Tröster, *Senior Member, IEEE*

Abstract

Context processing in Body Area Networks (BANs) faces unique challenges due to the user and node mobility, the need of real-time adaptation to the dynamic topological and contextual changes, and heterogeneous processing capabilities and energy constraints present on the available devices. This paper proposes a service-oriented framework for the execution of context recognition algorithms. We describe and theoretically analyze the performance of the main framework components, including the sensor network organization, service discovery, service graph construction, service distribution and mapping. The theoretical results are followed by the simulation of the proposed framework as a whole, showing the overall cost of using dynamically distributed applications on the network.

Index Terms

Simulation, analytical modeling and measurements of BAN, Integration of BAN with heterogeneous networks, Platforms for the support of BAN applications

C. Lombriser, D. Roggen, and G. Tröster are with the Wearable Computing Lab, ETH Zurich, Switzerland
R. Marin-Perianu and P. Havinga are with the Pervasive Systems Group, University of Twente, The Netherlands

I. INTRODUCTION

Body Area Networks (BANs) are composed of wireless nodes, ranging from hand-held devices such as mobile-phones, over smart objects in the environment to miniaturized sensor nodes integrated into garments. These devices provide a heterogeneous collection with varying capabilities in terms of sensors, actuators, processing power, memory, and available energy. The number and type of devices forming a BAN change over time, as a result of the interaction with other BANs, e.g. people exchanging objects, or between the BAN and the environment, e.g. clothes or objects taken from chairs.

A user-interface to such a system must hide the complexities involved in handling all the devices involved in the process [1]. One way of achieving this is to make the system context-aware, such as to infer the user state from a set of body-worn sensors [2]. Another type of context is the activity of the user, for which the Context Recognition Network (CRN) toolbox [3] provides a modular composition of algorithms for fast prototyping. Previous work has shown that miniaturized and low-profile sensor nodes with limited processing power are capable to run complex classification tasks, such as the recognition of sound [4].

In this paper, we focus on the organization of activity recognition on dynamic and resource-constrained BANs. We propose a flexible service-oriented framework, which provides the core-components for real-time adaptation to network mobility and heterogeneity. Network mobility is handled through dynamic, context-aware clustering and service discovery. The activity recognition is structured as a service graph, where the various services represent sensors, actuators, and miscellaneous data processing functions. The service graph is dynamically mapped to appropriate nodes in the BAN to execute the recognition algorithm. We model our approach for cluster stability, service graph executability and execution cost.

The paper continues with an analysis of the special properties of BANs and presents the characteristics distinguishing BANs from traditional networks in section II. We present a framework for the distributed execution of applications using the service-oriented approach in section III.

To provide a stable processing environment, where nodes stay interconnected, the network is clustered as described in section IV. A model of service distributions is introduced in section V to get an estimation on how many services need to be available on sensor nodes such that applications can be expected to be executable. Section VI introduces a cost model for applications mapped onto the network. This cost is then evaluated using simulations in section VII, showing the behavior and requirements on devices for successful deployment of applications in the BAN environment. The findings in the whole chain from network organization to application execution cost are discussed in VIII, whereupon the paper is concluded.

II. Characteristics of context-aware BAN applications

We give two examples of wearable smart assistants, in order to pinpoint the recurring characteristics of BANs and the challenges to distributed context processing. The first example is a generic personal sports trainer, while the second example is an assistant for industrial workers.

A. Personal sports trainer

In many sports, the effectiveness of the athlete can be improved by optimizing the course of movement. A BAN system can act as a personal trainer by monitoring the movements using sensors attached to different limbs of the body. It can suggest improvements or training sets to enhance agility or strength. Using actuators such as vibrators or sound, the system can also give feedback, helping the athlete to improve his motions.

As an example, we consider running tracks through forests, which are interrupted by stops with equipment, such as bars for pull-ups, or wooden dumbbells. The runners wear clothing with integrated motion sensors, sensing their whole body motion. When the runners reach a stop, they perform one of several activities available at this location, depending on their training targets: strength, agility, or endurance.

The BAN is able to automatically detect the sensors attached to the clothing and the ones the athlete interacts with. Depending on the activities to be performed at a certain stop, the BAN

downloads the corresponding service graph to monitor the runner's motion. The system must be able to distinguish among multiple runners and the various activities they are performing. For this application, a wide range of body-worn devices and accessories may be worn by athletes, requiring service graphs to be adaptable to many device types and configurations.

B. Worker training and support

A second scenario is the recognition of user activities at work [5]; workers in assembly manufacturing are supported with just-in-time context-aware information on the activities they are doing. Sensors in their clothing and tools measure movements to determine the activity currently performed. The BAN system can thus monitor whether all necessary steps were performed. If a mistake was made, the system may alert the worker in a suitable way.

Consider the example of two workers mounting the engine of a car. When the engine is suspended into the engine compartment, the BAN checks if one worker does not move the engine in such a way as to trap the other worker's hand. When they pick up the automatic screwdrivers, the tools are automatically checked for the correct settings, such as rotating speed or torque for the screws. The suspension system holding the engine cannot be released before both workers have completed screwing and stepped back from the car.

In this scenario, the BANs are dynamically adapted to incorporate the tools the workers are using. The system needs to distinguish among the nodes attached to different workers by correctly associating the tools with the corresponding BANs. The collaboration among workers becomes also a collaboration among BANs, which may connect to the company database and register completed process steps or issue emergency alarms.

C. Characteristics of a context-processing framework for BANs

From the two scenarios presented above, we derive the specific requirements of BANs, distinguishing them from typical Wireless Sensor Networks (WSN), and the resulting challenges for executing algorithms in these environments:

- **Heterogeneous devices** – BANs consist of devices with highly varying sensing, processing, and communication capabilities. This contrasts to the mostly application-specific and homogeneous networks generally considered for WSN research. As a consequence, a programming abstraction needs to be found which hides the details of the implementation and enables a programmer to write applications executable on a wide range of devices.
- **Dynamic topology** – As people move in the environment, BANs constantly connect and disconnect to other BANs, static devices or wireless networks. A first challenge lies in identifying clusters of nodes which retain communication for an extended amount of time. For achieving a stable distributed processing, a selection of nodes that can be expected to remain accessible with a high confidence is necessary.
- **In-network processing** – dynamic BANs require real-time in-network processing, delivering just-in-time feedback to the user. Sensor data is processed where it is sensed, and only the resulting events and alarms are transmitted for further context inference, therefore reducing the network load and at the same time enabling quick response times.
- **Continuous operation** – The timescale of events that need to be sensed on the body and its immediate surrounding is much shorter than that of the more common WSN deployed e.g. for crop field monitoring or building automation. Therefore, higher sample rates are needed (typically of the order of 10-100 Hz for human activity recognition from motion sensors). As a consequence, only limited time is available for duty cycling or entering sleep modes between data acquisition.
- **Energy resources** – Handheld devices and sensors integrated into clothes are in most applications removed from the body at night, which gives the opportunity to recharge devices during this time, e.g. using coat hangers in the wardrobe [6]. Energy resources must thus sustain a runtime of a few days rather than months or years.

A context processing framework designed for BANs must take into consideration these characteristics. Failure to do so is likely to result in poor or unpredictable performance in this environment.

We developed a service-based approach which takes into account the points highlighted above for efficient context processing in BANs.

III. Service-oriented context processing framework

In previous work, we have implemented the Titan framework for the execution of context recognition algorithms in BANs [7]. Applications within Titan are described using services and their dataflow interconnections. *Services* describe sensors, actuators, or data processing functions offered by members of the BANs. They are considered black boxes offering input and output ports, which can be interconnected to form a *service graph*, representing the application to be run. A service graph description includes parameters passed to the services and attributes constraining the assignment of services to network nodes.

Service graphs typically describe the dataflow from sensor readings to data processing result, such as shown in figure 1 for the recognition of motion activity. The service graph implements the algorithms used for the worker support example presented in [5]. The top row of services consist of motion sensors located on the body of a worker and on the tools he uses. The sensors produce input to the algorithm, while the next levels introduce the services selected for feature extraction and first local classification of the activity, which is then fused in the network for an overall result.

Each node in the network provides a set of services in a *service pool*, from which they can be instantiated when needed. An unused service does not consume RAM nor CPU cycles, such that the content of the service pool depends on program memory only, which is usually available in larger quantity than RAM. Devices with more resources may provide a larger number of services, and devices with different types of sensors may provide different sets of services in their service pools.

The distributed processing is organized as depicted in figure 2. Each network cluster contains a *Network Manager* responsible for the execution of applications, which are stored in their service

graph representation in a *Service Graph Database*. On a request for execution, the Network Manager retrieves the corresponding service graph from the database and passes it to the *Network Mapper*. The Network Mapper partitions the service graph in subsets and assigns them to network nodes for processing. Therefore it uses the information in the *Service Directory*, which maintains a database of node capabilities and service pools available on the cluster nodes. On the individual nodes, *Node Managers* are responsible of instantiating and executing the services of the service graph subset assigned to them.

Upon changes in the cluster, such as new nodes appearing or nodes going lost, the Network Manager is notified by the service directory, and reevaluates the mapping of the application service graph. Different error types are reported to the Network Manager as well and are forwarded to the initiator of the application execution request.

We want evaluate the behavior of the Titan framework in BANs. A first important process is the clustering algorithm, which organizes the network in which Titan is run.

IV. CLUSTERING ALGORITHM

The Network Mapper prefers instantiating services in the local cluster as it assumes those nodes will remain available with higher confidence than others. Thus the cluster should include all and only the nodes carried on the body of the user, which will be referred to as the *correct* state. We measure the *stability* p_S by the percentage of time the cluster is in the correct state.

To provide a stable cluster, network nodes are dynamically clustered according to a mutually shared contextual state using the Tandem algorithm [8]. Here, the shared contextual state is whether nodes are located on the same person, which can only be determined with a certain accuracy [9]. Due to this non-perfect accuracy, the perceived shared context may vary in time and lead to cluster instability. In order to analyze the influence of this instability on the service graph mapping, we evaluate the behavior of Tandem using the following mobility model for the BAN.

The BANs of individual persons are modeled by groups of sensor nodes moving around an area using a Random Point Group Mobility (RPGM) model [10]. When groups come into communication range of each other, they choose with probability p_m to move together to the next waypoint and to wait there for a random time. After the waiting period, they choose whether or not to continue with each other. This mobility model follows considerations of [11] to produce interconnectivity patterns resembling the social behavior of people.

We assume that the moments when the nodes start to cluster and when the clustering structure is freezed are triggered by a contextual change or a change in the environment. As an example, we use the moment when a person starts and stops walking or running, which can be reliably detected [12]. When the person is walking or running, Tandem permanently evaluates the shared context among the nodes wirelessly connected and distributively clusters the person's devices. Clusters may change depending on the perceived context (whether nodes are part of the same BAN) or as a result of topological changes (people meet or separate). At each time step, every node chooses itself or a neighboring node with which it shares a common context as *clusterhead*. Tandem tries to achieve stable clusters by keeping the same clusterhead as long as possible and by having each node make decisions based on the status of its neighbors with which it shares a common context. When the person stops walking, Tandem ceases to run and the clusters remain as they are.

As soon as a clusterhead is elected, it becomes the Network Manager and receives the service descriptions from each node in its cluster and stores it into its *service directory*. The Network Mapper can then query the service directory for services it needs to instantiate. In case services are not found locally, the queries are forwarded to service directories in adjacent clusters. Using service directories allows a quick assessment of the capabilities of a cluster and speeds up the mapping process.

As the next step, we model the distribution of services in the network. This distribution determines what services are available in each cluster and provide the solution space for the

Network Mapper.

V. SERVICE DISTRIBUTION MODEL

Titan relies on the services available in the service pools of the individual nodes in the cluster. This allows fast reconfiguration and quick adaptation, but raises the question of whether applications can be mapped to the network at all. Therefore we introduce a system model which allows deriving a probability of a service graph being executable in a cluster. Every sensor node has a subset S_a of all known services S available in its service pool. For the analysis we describe the probability of each service $s \in S$ to be available on an arbitrary node with probability $p(s) = \Pr[s \in S_a]$, with $p(s)$ being referred to as the service distribution. The availability, or the probability that a service is available on at least one node in a homogeneous network with n nodes, is:

$$a(s,n) = 1 - (1 - p(s))^n \tag{1}$$

Using equation 1, we can derive a general expression for whether a given service graph is executable in a network of with n nodes. A service graph $A = (T, I)$ is described by a set of services T and their interconnections $I = (t_i, t_j)$. Note that services $s \in S$ may be contained multiple times in T. We thus further introduce the set of *different* service $T_D \in T$ the service graph requires. The product of the service availabilities indicates whether all required services are available:

$$p_{exec}(n, T_D) = \prod_{s \in T_D} a(s,n) = \prod_{s \in T_D} (1 - (1 - p(s))^n) \tag{2}$$

The execution probability p_{exec} indicates whether a network with n nodes can provide all the different services T_D required to execute a service graph. It assumes that services can be instantiated multiple times at each sensor node and that the probability of a service being available is the same on every node. The execution probability can be adapted to heterogeneous networks as will be shown below.

The number of possible mapping solutions of service graphs to the network can be derived by calculating the expected number of nodes providing a service. As each of the nodes independently implements the service with probability $p(s)$, the expected number of nodes is following a binomial distribution having an expected value of $np(s)$. Titan allows to instantiate a service multiple times on every nodes, such that every service can be distributed independently on $np(s)$ devices. The number of possible mappings for a service graph is thus:

$$R(T,n) = \prod_{s \in T} np(s) = n^{|T|} \prod_{s \in T} p(s) \tag{3}$$

Services added to the service graph as well as the number of network nodes thus exponentially enlarge the number mapping solutions.

In order to further analyze the executability of service graphs on BANs and to show how the model can be adapted to a less abstract case, we analyze the special case where we assume no knowledge about what kinds of application service graphs should be run. We therefore distribute all services with uniform probability. However, we distinguish between powerful devices, such as PDAs or mobile phones, and limited devices, which are integrated into clothing and due to their minimal size can provide only limited processing power. Further we distinguish two classes of services, a set of simple ones $S_e \subset S$, which are available with probability p_{el} on limited devices, and with a probability $p_{ep} \geq p_{el}$ on powerful devices. The second set of complex services $S_i \subset S$, $S_i \cup S_e = S$, can only be implemented on powerful devices with probability p_{ip}.

Using the availability equation 1 and having a fraction r of limited devices in the network, we can derive the expected total number of different services $S_{tot}(n) \leq |S_e| + |S_i|$ available in a network with n nodes:

$$S_{tot}(n) = |S_e|(1 - (1 - rp_{el} - (1-r)p_{ep})^n) + |S_i|(1 - (1 - (1-r)p_{ip})^n) \tag{4}$$

During development, S_{tot} may serve as an indication of how large a service directory should be to accommodate all services available in the cluster. S_{tot} also allows to give a general probability for the executability for a service graph requiring $|T_D|$ different services, by considering the

ratio of all possible combinations of service graphs from the available services to the number of service graphs combinations from all existing services:

$$p'_{exec}(n, |T_D|) = \frac{\binom{S_{tot}(n)}{|T_D|}}{\binom{|S_e|+|S_i|}{|T_D|}} \tag{5}$$

p'_{exec} indicates whether the network can be expected to provide all the services required for execution. Figure 3 shows how p'_{exec} changes with the number of nodes and the size of the service graph. The factorials in the binomial coefficient are approximated for the continuous S_{tot} using the gamma function. A low number of different services improves the probability considerably. Adapting the service distribution to service usage statistics of service graphs intended to be run in the network can thus considerably improve p'_{exec} for service graphs following the statistic. It will howeverdecrease the range of different service graphs being executable.

The service distribution model delivers an executability measure for a service graph can serve as a guideline when designing the application service graph. However, during runtime more knowledge knowledge of the current state of the network is available. The actual cost in terms of resources required for execution can then be determined by the Network Mapper.

VI. Mapping services to network nodes

The Network Mapper assigns each service of a service graph to a node in the network and therefore needs to consider the availability of each service on the nodes as well as their resource constraints. Neither the processing nor the communication capabilities should exceeded on any of the nodes. Additionally, it is favorable to find the implementation keeping resource usage as low as possible.

The task of the Network Mapper is formally described as to map a service graph $A = (T, I)$ onto a network graph $G = (V, E)$. The network graph is described by a set of nodes V and communication links $E = (v_i, v_j)$, $v_i, v_j \in V$. The Network Mapper's goal lies in finding a mapping $M : T \rightarrow V$, such that a given cost function $C(M)$ is minimized.

Various cost functions targeting different trade-offs have been proposed for this task, such as the minimization of transmission cost, total energy consumed, or the maximization network lifetime [13]. In this paper we use a metric targeting minimization of the total energy used in the network. The cost function makes use of a model of the sensor node using values stemming from benchmarking the Titan implementation on sensor nodes [7]. The metric used for the evaluation relies on three main cost functions:

- **Processing cost** $C_p(t, v)$ – the cost of processing all services assigned to a node. This cost results into a measure for whether enough CPU cycles are available to execute all services of the subset assigned to the given node.
- **Sensor cost** $C_s(t, v)$ – the cost of using sensors to collect data for the algorithm. As sensors can usually be turned off when not sampling, this cost value describes the additional energy dissipated on the node while sampling, and includes possible duty cycling.
- **Communication cost** $C_c(i, v, e)$ – the cost of communicating data from one service to another for the node v. The communication cost is zero for two services communicating within the same node. For external communication, it prioritizes intra-cluster communication and introduces penalties for cross-cluster communication. The cost includes energy dissipated at the sending and receiving part. It is defined by the message rate.

The mapping is constrained by the maximum processing power $C_{p,max}(v)$ and communication rate $C_{c,max}(v)$ a node can support. These limits ensure the executability of the tasks on the nodes and guarantee that the maximum transmission capacity is not exceeded. The constraints are given for the service graph subset $(T_v, I_{v,e})$ assigned to a node $v \in V$:

$$\sum_{t \in T_v} C_p(t, v) \leq C_{p,max}(v) \tag{6}$$

$$\sum_{i \in I_{v,e}} C_c(i, v, e) \leq C_{c,max}(v) \tag{7}$$

Each interconnection i is mapped to an edge e and added to two sets $(i, e) \in I_{v,e}$ as outgoing and incoming connections. Failure in meeting the constraints results in the service graph not

being implementable. In such a case the execution cost will be set to infinity.

The total execution cost of the network is achieved by summing up all costs incurring at nodes participating in the execution:

$$C_{total}(M(A,G)) = \sum_{T_v \in T} \sum_{t \in T_v} C_p(t,v) + C_s(t,v) + \sum_{I_{e,v} \in I} \sum_{i \in I_e} C_c(i,v,e) \quad (8)$$

The costs introduce above depend on the device type to which they apply. The parameters for the device model are sent to the service directory along with the node address. The Network Mapper further uses a model of the services to derive the service output data rate given a certain input data rate and the service parameters in the service graph description. When determining execution cost, the Network Mapper first derives an estimation of the data communicated from service to service by propagating the data rates generated from each service to each successor. The individual cost functions make use of the service models and device models to produce the total mapping cost.

In this work we are interested in finding the absolute minimum mapping cost of the system. We do not have an algorithm to find an optimal service graph mapping with minimal cost in the current implementation of Titan. Also, an exhaustive search is intractable for service graphs and networks of reasonable size, as the search space grows with $O(n^{|T|})$ (see equation 3). Therefore we use a Genetic Algorithm (GA) to optimize the mapping, as GAs are known to provide robust optimization tools for complex search spaces [14]. The GA parameters are selected in order to favor convergence to the global maximum by selecting a large population size, avoiding premature convergence, and by performing several runs. The resulting performance is the maximum of the performance obtained in each runs.

The service graph is encoded for the GA as chromosome with $|T|$ genes. Every service is described by a gene containing all nodes in the network providing the service. Mutations are applied by moving services from one node to another. Crossovers arbitrarily select two

chromosomes, randomly pick a gene and swap the gene and all its successors between the two chromosomes, which are then added to the population. The fitness of the chromosomes is evaluated using the cost metric given above.

Once the implementation of the service graph with the lowest cost has been found, the service graph subsets can be sent to the individual nodes for execution. We stop our analysis at this level and do not further model the exact firing rules and scheduling problems on the individual nodes. In the next section, we simulate the whole chain of algorithms presented to this point and analyze their interactions.

VII. Simulation results

The interactions of the clustering algorithm, the service distribution, and the mapping are evaluated using the mobility model introduced in section IV. Simulations are run for 10'000 virtual seconds with 2 to 10 groups of 10 nodes and the probabilities $p_m \in 0.3, 0.5, 0.7, 0.9$ of groups joining each other for the next waypoint. The simulation provides an area of 100x100 units and a communication range of 5 units. The group speed averages at 3.62 units per second, while waiting times result in a mean of 10 virtual seconds. The accuracy of the shared-context recognition is modeled after experimental results that analyzed whether devices are located on the same person by correlating sensed motion patterns [9]. Figure 4 shows the stability p_S of the clusters depending on the number of groups n_S and the joint movement probability p_m. A higher value of p_m results in longer connections among clusters, introducing more opportunity for wrong shared context associations and hence lower stability. In a similar way, increasing the number of groups adds more confusion due to the higher number of possible clusters nodes can associate with.

Services are assigned to nodes in each simulation run following a uniform distribution as described in section V, with two device and service classes. The model parameters are set to $p_{el} \in [0.3, 0.4, 0.5]$, $p_{ep} = 0.7$, $p_{ip} = 0.3$, $S_e = 40$, $S_i = 20$, and $r = 0.8$. We repeat each

service distribution 100 times for each simulation run. This allows a statistical investigation of the execution cost of a given service graph according to the effective service distribution on individual nodes.

The execution costs are determined for a service graph resembling figure 1, with 10 instances of 6 different services, and costs requiring at least 3 nodes. The GA finds an optimal solution for each cluster configuration found in the simulation runs and each service distribution. The GA parameters constitute a population size of 150, a rank selection of the 90 best individuals, 6.7% mutation rate, and 50% crossover rate.

Figure 5 shows the average execution cost C_{total}, with its variance as shading, and the number of cluster nodes over time. Where the execution cost line is interrupted, no executable mapping could be found for any service distribution, indicating that the sink node receiving the result of the service graph execution is not in the cluster any more. Peaks in the variance result from powerful nodes having left the cluster, such that multiple low-profile nodes need to pick up the processing, leading to higher execution costs.

In figure 6 we represent the average execution cost of a service graph as function of p_{el} and r. In this case, we simulate two clusters, one of 5 and one external cluster of 20 nodes. We deliberately selected a cluster, whose size allows to illustrate the influence on the cost of p_{el} and the consequences of recruiting additional nodes external to that cluster. The variance in the cost is shown in figure 6b. With increasing p_{el}, the local cluster is able to support a larger part of the service graph, thereby reducing the cost due to the higher availability of powerful devices. A similar effect can be observed for decreasing r, which increases the number of powerful nodes and reduces the execution cost. The variances show the dependency of the cost from the actual service distribution. A high variance indicates that the cost can be substantially reduced for certain service distributions. The mean difference between maximum and minimum of the mean cost for different service distributions is 32%. This implies that there is a strong benefit for using statistical knowledge about the service graphs that might be issued onto the network.

By assuring a higher availability for often used services, the execution cost of these selected service graphs may be substantially reduced.

We validate our execution model by comparing the model-derived p'_{exec} to the experimental results observed in our simulation. We illustrate this in figure 7 by comparing p'_{exec} derived from the model and simulation for $r = 0.4$, $p_{ep} = 0.7$ and two values for p_{el}. Results do not match for low sensor number values, which is a result of the service graph not being executable on less than 3 nodes. For higher numbers of nodes ($n > 5$), however, the effect is reduced and the model closely matches simulation results. Therefore we conclude that our model can indeed be used as an indication of whether a service graph is executable on a network with uniform service distribution.

VIII. Discussion

Within this work we have identified the main characteristics for distributed context processing in BANs and have evaluated a service-oriented approach for its suitability to this environment. Several solutions for service-oriented processing in Wireless Sensor Networks (WSN) have been presented before, such as DFuse [13], OASiS [15], ATaG [16], TinySOA [17], or Viptos [18]. Those frameworks have been designed with different objectives and do not completely fit the requirements imposed by the applications we have presented.

The Titan framework is specifically designed for *in-network processing* of activity recognition algorithms. The challenge of *heterogeneity* encountered in BANs addressed by a service-oriented approach. Each service is defined by a standard functionality, expected input, and output produced. Hence, it can be implemented in native machine code for the individual devices and can still allow seamless (re-)programming, even in heterogeneous environments. Our framework allows choosing a subset of all services for implementation on participating devices, such that the service pool can be adapted to available processing resources.

Dynamic topologies are organized by our clustering algorithm to allow for stable execution

despite unpredictable dynamical changes in network topology. Therefore we make use of shared-context algorithms, which provide a more robust approach in comparison to simple monitoring of connectivity patterns. The clusterheads maintain service directories for quick access to their clusters' services from foreign clusters.

The minimization of resource usage for *continuous operation* is achieved by an appropriate cost model, which is used by the Network Mapper to assign services to nodes for execution. The ability to reprogram nodes allows saving energy at times when sensors and devices are not needed.

The general advantages of the service-oriented approach are e.g. the ease of programming. A programmer has just to interconnect services to create applications. No code has to be written and new services may be debugged individually, thus removing potential error sources from the development process. A compact service graph representation allows fast reprogramming of the heterogeneous network while native machine code implementations of services ensure efficient processing. Therefore the approach combines the advantages of virtual machines and code update mechanisms to provide an optimal solution for stream processing in sensor networks [7].

The results of this paper may be used by a context recognition algorithm designer to assess the requirements on service distributions and execution costs incurring in a BAN when developing his algorithm. Additional techniques, such as service composition [19], might allow adaptation of service graphs to the actual network configurations encountered at runtime. The system model developed here might serve as an opportunistic selection criterion for candidate service graphs. Therefore it needs to consider only a low number of network parameters, i.e. $p(s)$, S, and n, instead of requiring complete knowledge about available sensor nodes and services.

Titan and consequently also the model do not include the ability to migrate the execution of services from one node to another, such as e.g. DFuse allows it. By migrating the service state, i.e. its main variables, to a node providing the same service, nodes could locally improve the service mapping and thus optimize an initial mapping in a decentralized fashion [13]. This

however would require some form of standardized service state import/export format supported by all service implementations.

Another point not included in the model is the energy consumption over time. Techniques such as clustering sensors according to required classification accuracy and taking into account the energy consumed by the sensor nodes might prolong network lifetime by turning off unneeded sensor nodes [20].

The results of the simulations involving the whole chain from mobility model to service graph execution cost validate the model we have derived. The influence of the model parameters on the execution cost are shown for the special case of uniformly distributed services. This corresponds to the worst-case scenario for which there is no a-priori knowledge about the service graphs intended to be mapped onto the network. The results might lead to the conclusion that with a sufficient count of services on every node, service graphs are always executable and technological advancement in processing power density would give any kind of node the ability to hold enough services. With BANs however, it is more interesting to take advantage of this development by further reducing the size of the devices and by further integrating peripheral components. This would allow more unobtrusive ways of embedding devices into clothing, thus making the technology disappear.

IX. Conclusion

The detection of the context of a user enables the development of proactive and unobtrusive interfaces to complex networks composed of sensor and actuators integrated into clothing, hand-held devices, and smart objects in the environment. We have evaluated the characteristics of a service-oriented context processing framework for such dynamic Body Area Networks with respect to the distinctive challenges encountered in this type of network. Its capability of adapting to node mobility, topology changes, and heterogeneous processing resources provides a stable processing environment for applications in form of service graphs. The theoretical model and the

simulation results allow to configure a BAN system in such a way as to optimize the probability of application executability on a given service distribution.

The results allow designers of context recognition algorithms to estimate the requirements imposed by theirs algorithms on network size and configuration early in the design process. The model further provides a basis for heuristical optimization of service distributions for networks as well as service graphs.

ACKNOWLEDGMENT

This paper describes work undertaken in the SENSEI project (www.sensei-project.eu) supported by the European 7th Framework Programme with the contract number 215923.

REFERENCES

[1] T. Starner, "The challenges of wearable computing: Parts 1 and 2," *IEEE Micro*, vol. 21, no. 4, pp. 44–67, 2001.

[2] A. Krause, A. Smailagic, and D. P. Siewiorek, "Context-aware mobile computing: Learning context-dependent personal preferences from a wearable sensor array," *IEEE Transactions on Mobile Computing*, vol. 5, no. 2, pp. 113–127, 2006.

[3] D. Bannach, O. Amft, and P. Lukowicz, "Rapid prototyping of activity recognition applications," *IEEE Pervasive Computing*, vol. 7, no. 2, p. in press, 2008.

[4] M. Stäger, P. Lukowicz, and G. Tröster, "Power and accuracy trade-offs in sound-based context recognition systems," *Pervasive and Mobile Computing*, vol. 3, no. 3, 2007.

[5] O. Amft, C. Lombriser, T. Stiefmeier, and G. Tröster, "Recognition of user activity sequences using distributed event detection," in *Proc. 2nd European Conference on Smart Sensing and Context (EuroSSC)*, 2007, pp. 126–141.

[6] A. P. Toney, B. H. Thomas, and W. Marais, "Managing smart garments," in *Proc. 10th International Symposium on Wearable Computers (ISWC)*, 2006, pp. 91–94.

[7] C. Lombriser, D. Roggen, M. Stäger, and G. Tröster, "Titan: A tiny task network for dynamically reconfigurable heterogeneous sensor networks," in *Proc. 15. Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, 2007.

[8] R. Marin-Perianu, C. Lombriser, P. Havinga, J. Scholten, and G. Tröster, "Tandem: A context-aware method for spontaneous clustering of dynamic wireless sensor nodes," in *Accepted for: Internet of Things, International Conference for Industry and Academia*, 2008.

[9] J. Lester, B. Hannaford, and G. Borriello, "Are You with Me?" - using accelerometers to determine if two devices are carried by the same person." in *Proc. 2nd International Conference on Pervasive Computing (PERVASIVE)*, 2004, pp. 33–50.

[10] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communication & Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[11] M. Musolesi and C. Mascolo, "A community based mobility model for ad hoc network research," in *Proc. 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, 2006, pp. 31–38.

[12] G. H. Jin, S. B. Lee, and T. S. Lee, "Context awareness of human motion states using accelerometer," *Journal of Medical Systems*, 2007, Online First, DOI=10.1007/s10916-007-9111-y.

[13] U. Ramachandran, R. Kumar, M. Wolenetz, B. Cooper, B. Agarwalla, J. Shin, P. Hutto, and A. Paul, "Dynamic data fusion for future sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 3, pp. 404–443, Aug. 2006.

[14] D. E. Goldberg, *Genetic Algorithms in Search Optimization & Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[15] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits, "OASiS: A programming framework for service-oriented sensor networks," in *Proc. 2nd International Conference on Communication Systems Software and Middleware (COMSWARE)*, 2007, pp. 1–8.

[16] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, "The Abstract Task Graph: A methodology for architecture-independent programming of networked sensor systems," in *Proc. Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR)*, 2005, pp. 19–24.

[17] A. Rezgui and M. Eltoweissy, "Service-oriented sensor-actuator networks: Promises, challenges, and the road ahead," *Computer Communications*, vol. 30, pp. 2627–2648, 2007.

[18] E. Cheong, E. A. Lee, and Y. Zhao, "Viptos: a graphical development and simulation environment for TinyOS-based wireless sensor networks," in *Proc. 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 302–302.

[19] S. Kalasapur, M. Kumar, and B. A. Shirazi, "Dynamic service composition in pervasive computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 907–918, 2007.

[20] P. Zappi, C. Lombriser, E. Farella, D. Roggen, L. Benini, and G. Tröster, "Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection," in *Proc. 5th European Conference on Wireless Sensor Networks (EWSN)*, 2008, pp. 17–33.

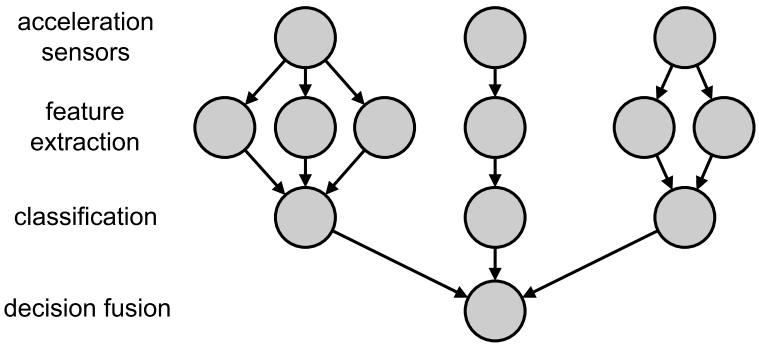


Fig. 1. Typical service graph for an activity recognition algorithm. The data flows from sensor services at the top through different data processing services to classify the activity observed by the sensors on the body of the user and the tools he uses.

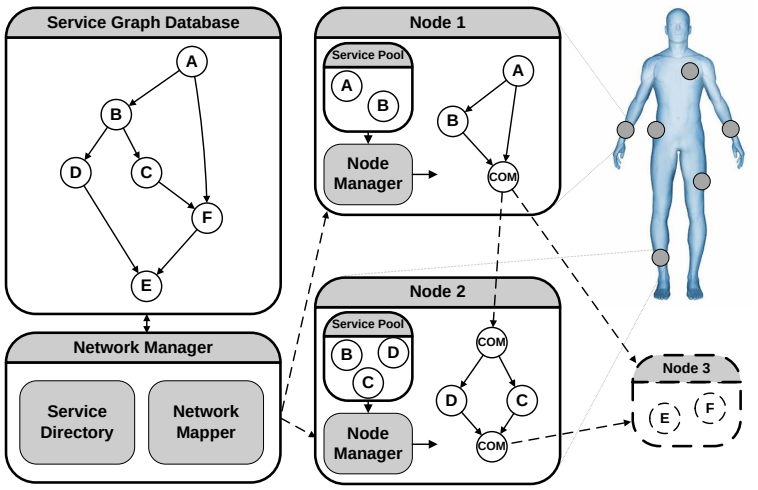


Fig. 2. Framework for the execution of service graphs. Each node can contribute to the distributed processing with the services in its service pool. The Network Manager assigns a part of the service graph to each node in the cluster to run the application.

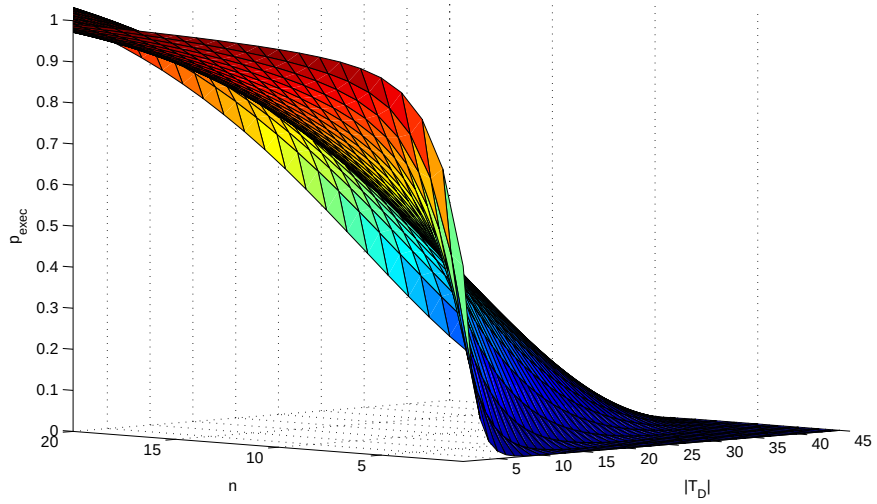


Fig. 3. Execution probability p'_{exec} of number of nodes accessible n vs. number of service types $|T_D|$ in the service task graph (for $p_{ep} = 0.8$, $p_{ip} = 0.4$, $p_{el} = 0.4$, $S_i = 5$, and $S_e = 40$). The execution probability increases with the number of available nodes n and decreases with the number of required service types $|T_D|$

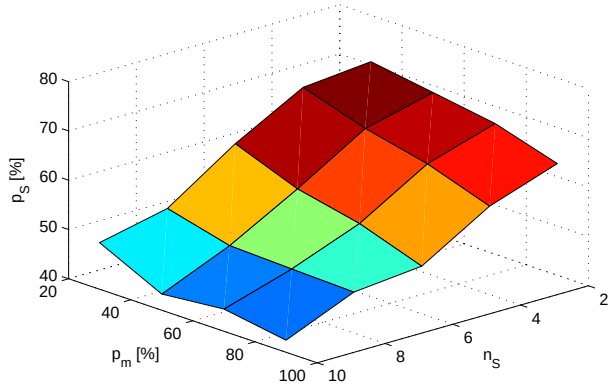


Fig. 4. Cluster stability p_S depending on the number of sensor groups n_S available in the simulation and the probability p_m of joint movement

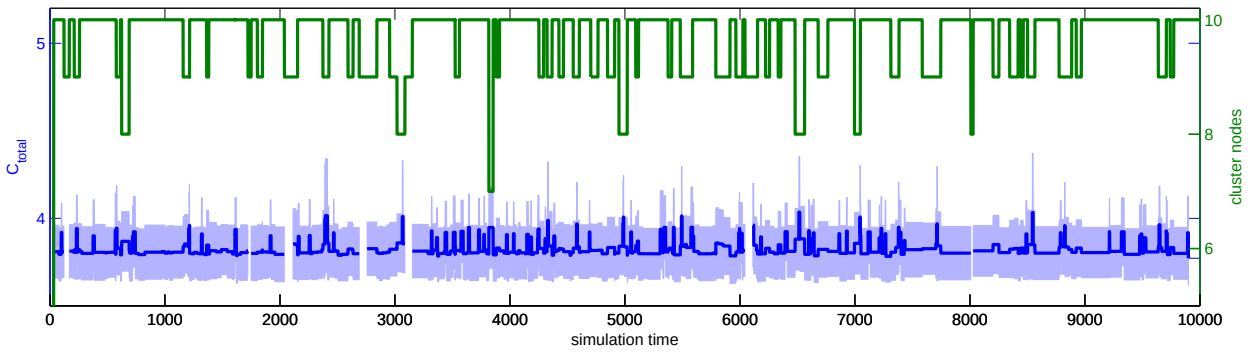
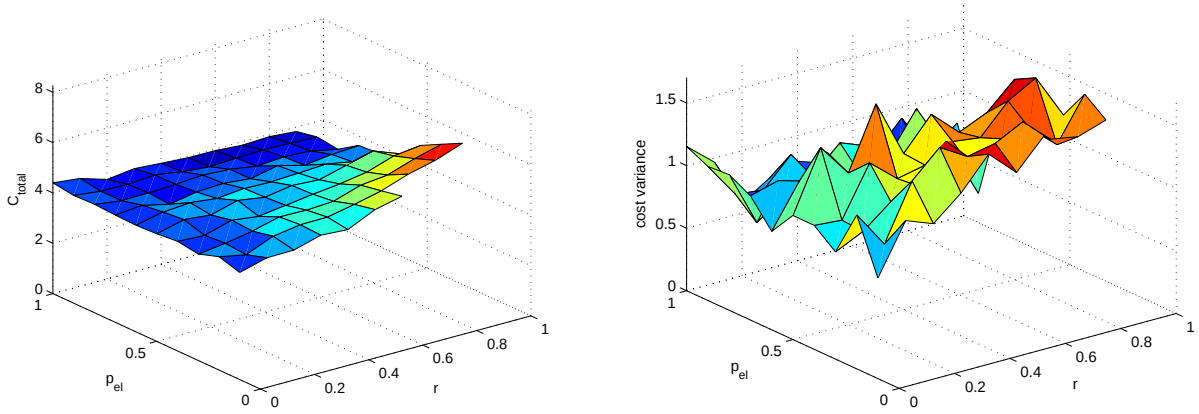


Fig. 5. Timeline showing on top the number of nodes available in the cluster and below the execution cost C_{total} with a shading indicating its variance for 6 node groups and $p_m = 0.9$. The clustering algorithm mostly only looses single nodes and has a stability of $p_S = 0.5$.



(a) Mean cost for the execution of the example service graph

(b) Variance of the cost over different distributions

Fig. 6. Execution cost C_{total} for a service graph with $|T_D| = 10$ on a 5 node cluster connected to a 20 node cluster. With increasing r, more low-profile nodes are in the cluster, requiring higher cost. With increasing p_{el} more services are available on all nodes, allowing a cheaper mapping. High variance indicates large cost differences for different service distributions. Application-specific service distributions can improve cost especially in regions with high variance.

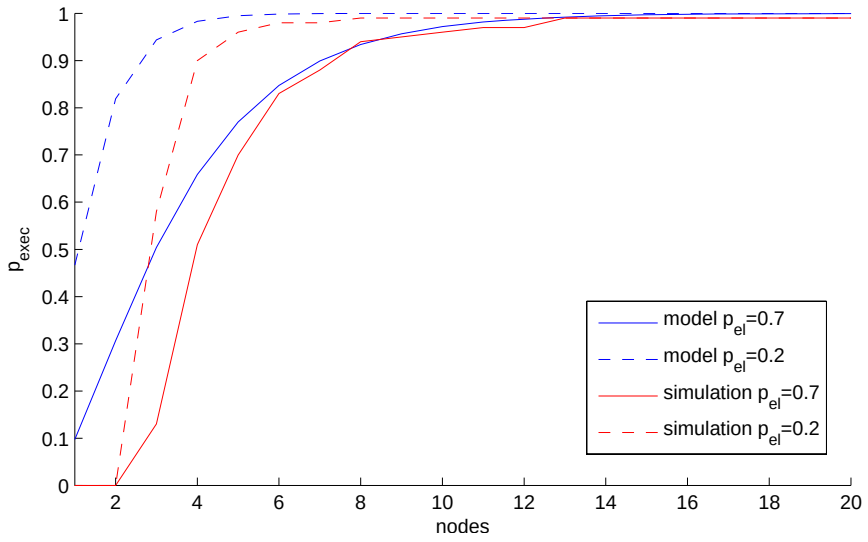


Fig. 7. Assessment of the model validity for the execution probability p'_{exec} for a range of nodes for a service graph using $|T_D| = 6$ service types, $r = 0.4$ and $pep = 0.7$. The model does not include execution costs, which result in the simulation values not reaching the model prediction for a low number of nodes, where costs invalidate most of the possible solutions.