Autonomic Role and Mission Allocation Framework for Wireless Sensor Networks

Themistoklis Bourdenas[1,2], Kenji Tei[2], Shinichi Honiden[2], Morris Sloman[1]

Department of Computing, Imperial College London, UK [1]
National Institute of Informatics, Tokyo, Japan [2]
{*t.bourdenas07, m.sloman*}*@imperial.ac.uk,* {*tei, honiden*}*@nii.ac.jp*

***Abstract*—Pervasive applications incorporate physical components that are exposed to everyday use and a large number of conditions and external factors that can lead to faults and failures. It is also possible that application requirements change during deployment and the network needs to adapt to a new context. Consequently, pervasive systems must be capable to autonomically adapt to changing conditions without involving users becoming a transparent asset in the environment. In this paper, we present an autonomic mechanism for initial task assignment in sensor networks, an NP-hard problem. We also study on-line adaptation of the original deployment which considers real-time metrics for maximising utility and lifetime of applications and smooth service degradation in the face of component failures.**

***Keywords*-task allocation; wireless sensor network; self-adapting; autonomic**

I. Introduction

The vision of pervasive applications is to blend in the users' environment seamlessly and interoperate transparently with them. However, such systems include physical components which may fail, sensors that change characteristics due to chemical fouling or the need to adapt to changing user context or behaviour [1]. Ideally, a system should cope autonomically with these changing conditions and adapt to failures, reallocate or deploy functionality without interrupting the user experience.

A typical pervasive application involves the deployment of a variety of tasks on a number of heterogeneous sensor nodes that cooperate to provide a service. The nodes may differ with respect to processing power, hardware resources, sensor devices, storage and communication capabilities. An allocation process should consider such attributes in order to meet the requirements of each task and maintain the quality of the service at high levels.

Autonomic computing [2] brings together many disciplines in order to build self-managing systems that can adapt to user needs and their operating environment. There have been several approaches to systems being capable of adapting to changing conditions without manual administration [3]. Autonomic computing is characterized by properties of self-configuration, self-healing/protection and self-optimisation. A closed feedback control-loop with four steps – *monitoring*, *analysis*, *planning* and *execution*, as illustrated in figure 1, is commonly present in such systems.

Monitoring tracks performance metrics from sensor components incorporated in the system to determine the condition of different parts of the network. The *analysis* phase processes information collected in order to infer system status. Analysis of monitored system attributes may indicate node failures, faulty sensors, poor or failed communication links, low battery levels as well as poor quality of service due to overloaded processors or communication links. The *planning* phase decides what actions are necessary, if any, to transition to an optimal state given the symptoms that have been identified by the analysis phase. Task migration is a potential adaptation mechanism in pervasive systems in order to cluster closely interacting tasks, reduce network traffic, prolong battery life of critical nodes or respond to component failures. A dynamic reallocation service must be incremental considering overheads of task migration – in essence, most of the network's state should be maintained unless there is a critical failure that requires more significant system restructuring. Finally, the *execution* phase applies the reconfiguration plan that has been produced, which typically requires an infrastructure that enables in-situ dynamic system reorganisation without disrupting its operation.

In this paper, we focus on the *planning* phase and more specifically on dynamic task reallocation in pervasive systems describing how we integrate this mechanism in the Starfish framework for autonomic pervasive systems. Typi-

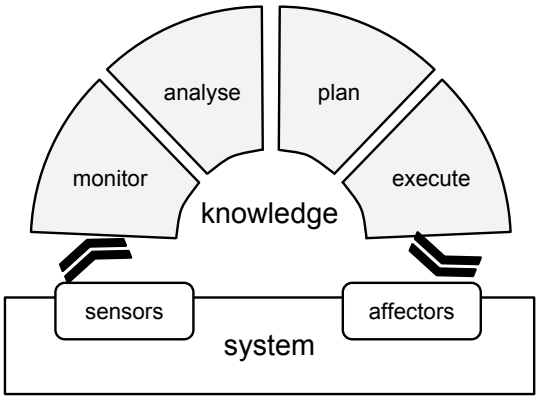


Figure 1. Closed feedback control-loop in autonomic systems

cally, task allocation approaches aim for a network set-up with minimal energy consumption or prolonged lifetime [4]–[8]. However, our focus lies on self-healing aspects of the network, responding to component failures or degradation. Consequently, we try to balance economical allocation and system reliability. This implies both high delivery rates among nodes as well as good quality of the transported sensed data. Unlike other approaches, we do not limit nodes to a single task. In event-based programming that is predominant in pervasive and sensor network applications, a node is usually assigned several short-lived, periodic tasks to execute concurrently. Furthermore, we describe how system requirements and constraints are extracted directly from the application specification in our framework.

In the following section, we provide a motivating example for the study of the task allocation in a pervasive system. In section III, a brief overview of the Starfish platform is presented as background for the mission framework. Section IV lays out the formal definition of the problem and defines it as an Integer Linear Programming (ILP) problem, while section V relaxes the requirements describing a heuristic approximation for the NP-hard combinatorial problem, using the simulated annealing algorithm. In section VI, we provide simulated experimental results on the effectiveness of the allocation mechanism. Related work in the area is presented in section VII and we, finally, conclude in section VIII.

II. A Healthcare Scenario

We consider a scenario from health-care, where patients' condition is monitored by a network of wearable sensors that monitor patient attributes, such as temperature, blood pressure and ECG, as well as his activity, using a 3D accelerometer. Reports are sent to a terminal node that can produce alerts for the medical staff of a clinic.

Figure 2 illustrates the application's specification as individual tasks that compose the service and their interactions. The application uses three thermometers and accelerometers tasks for enhancing sensing accuracy. Temperature and acceleration fusion tasks collect readings from individual sensing tasks to extract a single reading for the monitored attribute. Accordingly, a *Vitals Fusion* task collects extracted features to make a decision on the patient's state based on temperature, ECG signal and blood pressure. Another branch of the application performs *Activity* recognition from accelerometer readings. The final decisions are collected by the *Infer Status* task that makes a decision on the patient overall status given his context. If necessary, an alert is triggered from the *Alert* task.

Arrows between tasks denote the flow of information in the task graph. The application exhibits a hierarchical structure with multiple information fusion centres. Typically, leaf nodes are sampling tasks that encapsulate sensing devices of the network, while intermediate nodes are fusion and processing centres. Even though, for simplicity, the example

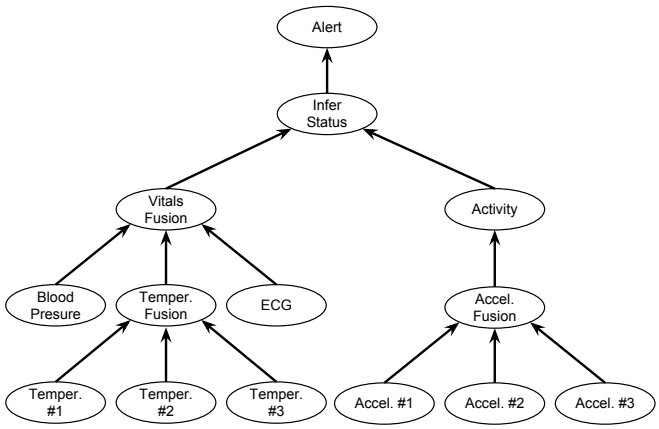


Figure 2. Healthcare scenario: patient monitoring task graph

application is a tree graph, our approach can also be applied to graphs that are not necessarily acyclic or may involve several sinks.

It is evident that tasks have different hardware requirements, which constrains their placement in the network. For example, the *ECG* task must be placed on a node with an appropriate ECG sensor. Moreover, if there are multiple available sensors the most reliable and accurate, should also be the most desirable. Another consideration for placement would be to minimise the communication cost between nodes by clustering cooperating tasks. This can either be achieved by being in the same node, thus eliminating the cost or by placing them in nearby nodes with good communication channels. Finally, we must also consider that certain tasks are incompatible, and cannot be placed together or that some tasks need to be placed only in specific locations. Example constraints are the three *Temperature* tasks should be placed on different sensor nodes to achieve redundancy or that at least one *Accelerometer* task needs to be on a node attached the patient's arm.

Even for simple applications, this task assignment to nodes can be overwhelming for a human administrator. The process does not scale well with the size of the network and it becomes impossible to handle manually in a dynamic environment, where components degrade or fail and response time is critical. Processes and tools are necessary that will allow the definition of application requirements for autonomic orchestration and deployment that reacts to component degradation and failure during system's lifetime.

We assume the existence of a monitoring and fault-detection mechanism that collects information from the network to determine the system status. The description of such mechanism is beyond the scope of this paper. We have previously studied fault-detection approaches [9] that can help identify the condition of sensors on different nodes. Here, we focus on the planning phase of the closed-feedback loop, which constructs a reaction plan based on

collected observations. We consider that multiple tasks can be executed on a single node concurrently. Typically, tasks in such systems perform short chunks of processing, e.g. sensor sampling, data buffering, feature extraction. The programming paradigm is, typically, event-driven, i.e. tasks are triggered by a network message, timers, readings, etc. Most of the studies in traditional distributed systems for task allocation consider tasks as long-running processes that nodes run exclusively. However, we argue that this model does not apply to pervasive applications.

In the following section, we introduce the Starfish framework and demonstrate how a scenario like figure 2 can be expressed in Starfish, extracting automatically the task graph and most of the constraints described above directly from the specification.

III. Starfish Framework

The Starfish framework [10] supports development of adaptive, autonomous applications for power constrained, embedded devices running the TinyOS 2.x operating system. It is a realisation of the Self-Managed Cell (SMC) architecture [9] that incorporates the basic building blocks to form federations of autonomous components that compose to large-scale autonomic systems. Starfish entails an embedded policy evaluation and enforcement system, *Finger2*, a *library* of high-level modules and a *desktop client* that assists application authoring.

The main software components of an SMC, as shown in figure 3, are an *Event Bus* that handles delivery of messages between components, a *Discovery Service* to detect new wireless nodes with embedded resources and include them in the cell. The *Finger2 Policy Management* system for Starfish facilitates the configuration and adaptation mechanisms to orchestrates the interactions among participating components within the SMC. In essence, every node in the network is an autonomous SMC and creates federations with other SMCs to form an interacting system. The grey boxes in the figure are additional components and services that provide self-healing properties in the autonomous cell and correspond to the four phases of the closed feedback control-loop.

A. Starfish Abstraction Mechanisms

The main abstraction mechanism in Starfish are the *policies*, simplified from those in Ponder 2 system [11], that are enforced on nodes. *Obligation policies* are event-condition-action (ECA) rules, used for configuration of the system allowing behavioural adaptation of the network, without the need to reprogram node images over-the-air. They can be considered a form of event-based scripting for nodes. *Authorisation policies* specify access-control rules for resources allowing to express permitted interactions between remote components. We focus on obligation policies as they are the fundamental adaptation mechanism in the framework.

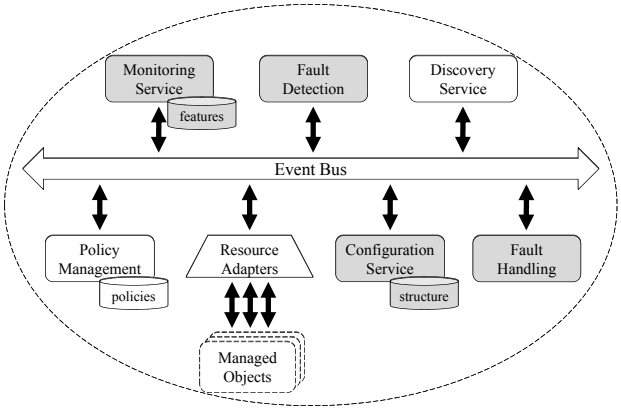


Figure 3. Self-Managed Cell architecture

Obligation policies operate on modules deployed on nodes. *Modules* are software components that encapsulate hardware (e.g. sensors, network interfaces, etc.) or provide functionality, such as mathematical functions or storage buffers. They are, essentially, the resources available on nodes. They abstract the low-level operations of nodes and provide functions that can be invoked or events that trigger execution of policies. Starfish provides a set of modules that are commonly used in WSN application, such as sensing, buffering, feature extraction, communication, etc. Obligation policies are triggered by events that originate from software *modules* or remote events that are received from the network, i.e. other nodes with appropriate credentials. If the policy condition predicate is true, a set of actions is performed. Both condition predicates and actions are functions that are provided by *modules*. Even though syntactically similar, predicates differ from actions by not involving side effects, such as modifying the state of a module or triggering events.

Policies can be defined in terms of *Roles*, e.g. temperature sensor, ECG sensor, acceleration fusion etc., which can be considered placeholder references which are then assigned to nodes at runtime when the actual deployment is realised. Nodes are associated with roles and each role generally performs a specific task. This enables policies to be authored without knowing the specific physical nodes within the distributed applications.

A *mission* is specified as a deployment of a set of policies, pertaining to an application, over the network infrastructure. For instance, monitoring of the average temperature in a building can be described as a set of policies that collect readings to local cluster-heads, analyse them and periodically report to the network sink on the condition in their corresponding area. Aggregation of policies into *missions* facilitates the management of an application.

A *configuration* is the initial set-up of an individual node. This includes the code to support *modules*, and the set of *missions* and *roles*. Loading of *modules* is not dynamic, i.e.

```
def Accel1
    on timer.Off(type)
    if type is ACCEL
    do AccelFusion.fusion.Collect(
            LOCAL_ID, type, accel.Sample())
```

Periodically sample acceleration and send it to the fusion centre, which correlates reading from multiple sensors.

```
def AccelerationFusion
    on fusion.Ready(type, value)
    if type is ACCEL
    do ActivityRecognition.activity.Update(
            type, value, timer.Now())
```

When the fusion module has a value ready, it is forwarded to the activity recognition component.

Figure 4. Health-care scenario in policies

a node cannot modify its modules during runtime. Instead, they need to be halted re-flashed with a new image and rebooted. This is a technical limitation of the operating system, TinyOS 2.x. While approaches for removing this limitation have been studied [12] [13], they involve significant overheads in battery consumption for transmission of new images, as well as disruption of operation to reboot the node. Starfish uses the modular structure of TinyOS 2.x loading only those modules that are necessary, as ROM space is typically constrained to a few hundreds of kilobytes.

However, *policies* are interpreted so*missions* and *roles* can be dynamically loaded or re-assigned at runtime, provided that required modules are pre-loaded on a node. Starfish provides mechanisms (modules) that manage dynamic role assignment and mission re-deployment.

B. Mission Specification

Starfish provides the abstractions required to support flexible dynamic adaptation for pervasive applications to overcome the limitations of TinyOS. In this section we describe how actual deployment of the specification can be automated by the system by illustrating how we can construct task graphs from obligation policies and attached requirements.

Figure 4 demonstrates a couple of policies that define tasks as presented in the healthcare scenario of figure 2. The first policy, *Accel1*, belongs to the task that samples patient's movement using accelerometers. The second is part of the *Acceleration Fusion* task. Tasks in the graph correspond to *starfish roles*. In this example, the tasks are simple enough that each role contains a single policy.

Policy *Accel1* is triggered by a scheduled timer event for sampling acceleration. The event is generated by the *timer* module, encapsulating the node hardware timer. The policy subsequently invokes the *Collect()* function from module *fusion* that is associated with another role, *AccelFusion*, in the application. It should be noted that the role *AccelFusion* may be assigned at run-time to the same node, resulting to a local invocation. Whether an invocation is local or remote is abstracted from the application developer and handled by the Starfish framework, hiding the underlying complexity of communication.

Finger2 policy action invocation syntax contains three components. The first component is optional indicating the role that the call is referencing. If omitted, the call is translated as an invocation local to the node. The second part indicates the module the action belongs to. The last part of the call is the operation to be invoked plus actual arguments, which can be constant values, such as the local node id ($LOCAL\_ID$), context variables of the fired event, such as the timer identifier ($type$) or values obtained from local function calls, such as the reading from the accelerometer ($accel.Sample()$).

Accordingly, policy *AccelerationFusion*, which belongs to the *AccelFusion* role, is fired when the module *fusion* emits event *Ready()* with a new value generated from collected accelerometer readings. The fused value is used to update the *ActivityRecognition* role for inferring user activity.

Figure 2 illustrates how obligation policies can be used to script interactions between functional roles in the system and build pervasive applications using available resources. Given a set of missions and their associated roles, we can extract a task graph, which indicates dependencies between roles and resource requirements, directly from policy analysis. Resource requirements can be determined by collecting modules referenced in policies. Thus, it can be determined that policy *Accel1* requires a *timer* and an *accelerometer* module on a node to run. The *Discovery Service* of the SMC can match these requirements with the directory of resources available in nodes.

Policy analysis provide adequate information for automating the role assignment process. However, one property that is not extracted from analysis is separation of tasks. For instance, it is not feasible to reason directly from policies that the temperature tasks need to run on different nodes for redundancy. Consequently, a developer must explicitly indicate such constraints as mutual exclusion between missions to be allocated to different nodes.

In the remainder of the paper, we describe an allocation service that performs the initial deployment of roles and missions in a wireless sensor network. Furthermore, it dynamically updates the initial plan during operation, when performance metrics indicate that the current configuration is no longer optimal in order to cope with failing components.

IV. Integer Linear Programming Formulation

The task of role allocation in nodes is a combinatorial optimisation problem, reduced to a generalization of the Quadratic Assignment Problem (QAP), an NP-hard problem, where n facilities need to be allocated to n locations minimising the cost of allocation given a flow between facilities and distances, i.e. weights, between locations. There are no co-location constraints in QAP, only a single separation constraint.

We formally define the mission allocation problem in Integer Linear Programming (ILP). This entails mapping the mission graph $G_t(T, E)$ and its constraints, extracted from policies, to the network graph $G_n(N, L)$. The problem is similar for both the initial and dynamic allocation that occur during the lifespan of an application.

For the graph node sets T and N, of G_t and G_n, respectively, a set of binary variables $x_{t,k} \in X$ is defined, where $t \in T$ and $k \in N$. If variable $x_{t,k}$ is set then task t is allocated to node k. Consequently, the objective function that we need to maximise is given by equation 1.

$$\epsilon = \sum_{\substack{t \in T \\ k \in N}} f(t,k) x_{t,k} + \sum_{\substack{i,j \in N \\ e=(a,b) \in E}} g(e,i,j)(x_{a,i} \cdot x_{b,j}) \quad (1)$$

There are two main components in the equation. The first sum is the utility function $f(t, k)$ for allocating a task to a node. The second sum is the utility function $g(e, i, j)$ that represents the cost of allocating two tasks a, b that are dependent in the mission graph $G_m(T, E)$, in two nodes i, j of the network. It is possible that $i = j$.

In addition to the objective function, ILP allows the definition of constraints on the allocation problem. We identified three constraints in section III. The first is the unique allocation of a task, expressed in equation 2. Every task instance can only be placed on a single node. Note that in the example of figure 2 the leaf temperature and acceleration sampling tasks are multiple instances of the same task type.

$$\sum_{k \in N} x_{t,k} = 1, \ t \in T \quad (2)$$

The second constraint discussed was the separation of tasks to prevent placement on the same node. For a set T_s of mutually exclusive tasks, the constraint in equation 3 enforces that they will be spread out appropriately. For instance, in the healthcare scenario such a set of mutually exclusive tasks would be the three temperature sampling tasks, so as to use different thermometers for increasing readings redundancy.

$$\sum_{t \in T_s} x_{t,k} \leq 1, \ \forall k \in N \quad (3)$$

Finally, there are cases that some of the tasks need to be constrained only to a specific area of the network. For instance, we need to measure acceleration from the sensors attached on the patient's torso, not his arm. In such cases a set N_c of eligible nodes can be defined by the policy author and enforce the property with the constraint in equation 4 in addition to constraint 2.

$$\sum_{k \in N_c} x_{t,k} = 1, \ t \in T \quad (4)$$

Having defined the objective function and the relevant constraints, the node and communication utility functions should be defined next. The utility functions express the properties of the selected allocation. We consider the accuracy and availability of the required resources, i.e. sensors, as well as the packet drop rates of communication channels between nodes selected to perform interdependent tasks to achieve high quality and reliable measurements.

The node utility function, defined in equation 5, concerns the quality of resources on a node. As explained earlier, resources specific to a node are identified by the Starfish modules. For resources such as a sensor, an associated score is attached that is indicative of the quality condition of the node. The specifics of the monitoring service that extracts the condition of sensors and applies a scoring scheme have been discussed in previous work [9].

$$f(t,k) = e_k \prod_{r \in R(t)} score(r,k) \quad (5)$$

While we are trying to achieve the best quality readings, energy remains a consideration and is also part of the utility function, favouring nodes with higher residual energy levels. Similarly, the communication utility function, in equation 6, considers $DR_{i,j}$ the message delivery rate between nodes i and j that is measured by the monitoring service of the network. Nodes are not necessarily directly within range and may have to relay messages via intermediaries.

$$g(d,i,j) = min(e_n) \cdot D_d \cdot DR_{ij}, \ \forall e_n \text{ and } n \in L_{i,j} \quad (6)$$

$L_{i,j}$ is the set of nodes that relay a message from node i to node j. The minimum residual energy among the nodes in the path is selected in order to avoid depleting individual nodes in the network.

The original objective function in equation 1, is not linear due to the $x_{a,i} \cdot x_{b,j}$ product. However, there is a standard way for transforming it into a linear one by introducing a set of binary, utility variables Y that replace the product in the equation.

$$y_{a,i,b,j} = x_{a,i} \cdot x_{b,j}, \ y_{a,i,b,j} \in Y \quad (7)$$

In addition, some new constraints are required that maintain the properties of the replaced binary product, as presented in equations 8-10.

$$y_{a,i,b,j} - x_{b,j} \leq 0 \tag{8}$$

$$y_{a,i,b,j} - x_{a,i} \leq 0 \tag{9}$$

$$x_{a,i} + x_{b,j} - y_{a,i,b,j} \leq 1 \tag{10}$$

The simplex algorithm, used in LP-solvers to find the optimal solution, tries to minimise the objective function, instead of maximising it as in equation 1. There is again a straightforward transformation of the function to achieve this.

V. Heuristic Approximation

The simplex algorithm in linear programming problems gives an optimal solution in polynomial time. However, when the variables are constrained to be integers, as in the formulation above with the binary $x_{t,k}$ variables, the problem becomes NP-hard. Consequently, the solution is not viable when the network size or the complexity of the mission increases. A heuristic approach for selecting a viable solution is described in this section. The problem is in essentially a combinatorial optimisation selecting T binary variables from a set of $T \times N$ variables. The solution is a combination of those variables with some additional constraints that were discussed before. An exhaustive exploration of all combinations requires exponential complexity of $O(2^{T \cdot N})$.

Typically, heuristics involve either random sampling of solutions or stepwise improvements on a selected solution. The former is inefficient as it does not provide any gradual improvement over-time. The latter includes hill-climbing approaches, where the algorithm tries to make small changes to a randomly selected solution to transition to a better one. This method is usually trapped in local optimum solutions when the problem space is not convex.

Simulated annealing [14], shown in figure 5 is a general meta-heuristic algorithm that combines the two approaches. It tries to emulate atoms' behaviour during the cooling of metal materials that try to reach a balance state by making jumps and releasing energy. Jumps are more frequent on high temperatures. As the material cools down, the jumps get less probable and the system, as a whole, reaches an equilibrium state.

Similarly, simulated annealing includes a temperature parameter, K that decreases over time with a rate $c \in (0, 1)$. Like hill-climbing the algorithm tries to transition to a better solution, one with lower cost $C(S)$, by making small adjustments to the existing one. While the temperature is high, there is a relatively high probability that the algorithm will accept as progress a solution with a higher cost than the current one. Consequently, this will eventually allow it

$K \leftarrow 1$
$S \leftarrow$ random solution
repeat
  for $i \leftarrow 1$ **to** M **do**
    $S_i \leftarrow$ transition(S)
    if $C(S) \geq C(S_i)$ **OR** $rand(0,1) < e^{\frac{C(S)-C(S_i)}{K}}$
      $S \leftarrow S_i$
  $K \leftarrow c \cdot K$
until $C(S)$ does not change

Figure 5. Simulated annealing algorithm

to escape from a local minimum to another area. As the temperature drops, it becomes less likely to make such jumps and eventually the termination condition is reached.

Frequent transitions at the initial rounds allow the algorithm to search a larger problem space until it, finally, reaches an area that will explore exhaustively. The approach again does not guarantee an optimal solution, but in practice it gives a very good estimation, having explored a broad spectrum, in relative short time.

We apply the simulated annealing process looking for the permutation of vector $V = \{x_{t,k} \mid t \in T, k \in N\}$. The cost function of a solution is derived by equation 1 in the original ILP formulation by transforming the utility to cost functions. The algorithm is making transition by randomly switching a variable, i.e. a task allocation, in vector V in every step. A transition is first asserted against constraints 2-4 before being considered as an eligible solution. It should be noted that the objective function does not need to be transformed to a linear equation any more.

The complexity of the heuristic is not directly dependent on the input and is capped by the parameters of steps and rounds of the algorithm. While there is no time guarantee, in practice it only takes a few rounds for the algorithm to stabilise and provide a solution.

VI. Evaluation

We present experimental results for the effectiveness of our approach to autonomic dynamic reconfiguration of the network. Initially, we consider the ability of the simulated annealing heuristic to approximate the optimal solution. Later we attempt to quantify the benefit of the reconfiguration in terms of sensor data quality and packet delivery and, finally, we compare the life-time performance of our approach to more power-saving focused methods that appear in the literature. For the evaluation, we use a realistic sensor network application specification, as was described on section II. In addition, we use larger examples of randomly generated task graphs, in order to investigate how the approach scales on larger networks and missions.

A. Quality of Solution

Initially, we examine how close the heuristic solutions compare to the optimal. For the ILP problem solving, we

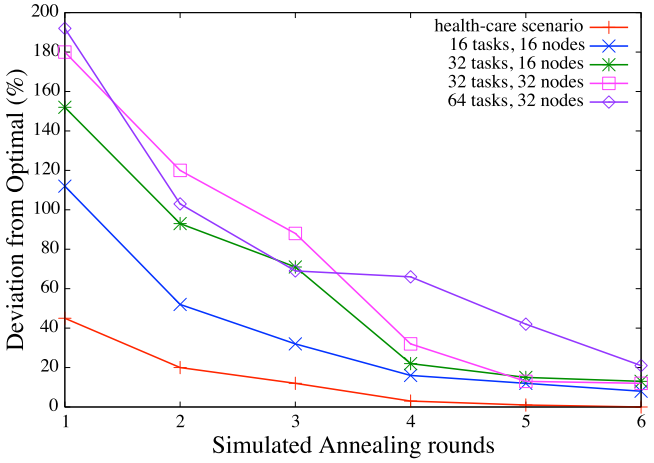


Figure 6. Simulated annealing approximation quality

used the GNU Linear Programming Kit[1], while for the simulated annealing we implemented a prototype in Python. Even though simulated annealing runs as an interpreted language, the execution time is in the order of a few seconds per round, while the execution time in the native implementation of the ILP solver requires several hours or days for large mission sets.

Initially, we map the health-care scenario on a wearable network composed of eight nodes. Each node is equipped with a different set of sensors, so that there are four thermometers, four 3D accelerometers, two ECG and two blood pressure monitors in total. We, further, use randomly generated task and network graphs, where for each sensor a random score is allocated and the network is connected with asymmetric links with random packet drop rates.

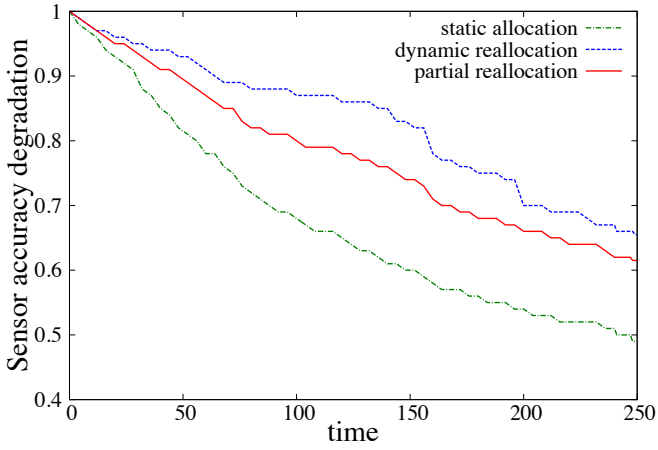
Figure 6 compares the solutions provided by simulated annealing in relation to the optimal solution. In the custom scenario of 14 tasks and 8 nodes, simulated annealing usually reaches the optimal solution only in a few number of rounds. Similarly, we try the same test for randomly generated graphs of different sizes and we observe that the simulated annealing can approach very close to the optimal solution, typically in 5 to 6 rounds.

We limit the experiments to problems with 64 tasks and 32 nodes as computation time for the ILP explodes and calculation becomes very lengthy. However, the trend demonstrates that simulated annealing is capable of providing similarly good solutions as the problem size increases.

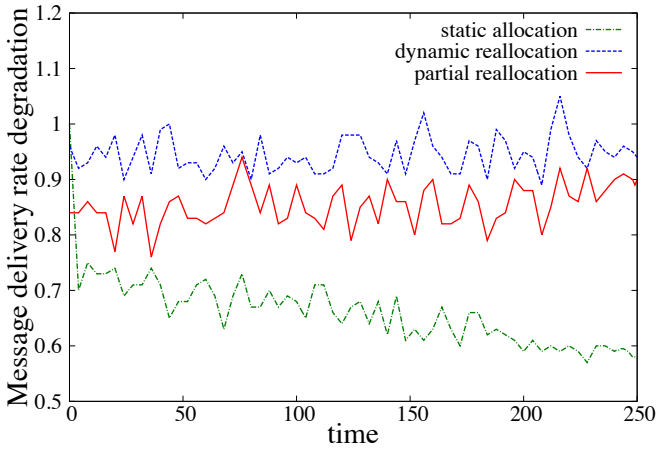
B. Quantify Benefits

We, next, quantify the benefits of dynamic adaptation and task reallocation at runtime. Runtime reallocation implies transmission of tasks over the network. In the case of Starfish, this is a relative cheap process, as tasks, which

[1]http://www.gnu.org/software/glpk/



(a) sensor accuracy degradation



(b) message delivery rate degradation

Figure 7. Quality degradation in the network

are encoded in compiled obligation policies are typically between 20-100 bytes. However, in other frameworks task migration might involve transmission of much larger binary images for nodes. Consequently, constant relocation of tasks can be inefficient or even impractical for many platforms.

We consider a more conservative reallocation scheme to reduce this overhead. A partial reallocation of tasks, whose quality metrics have dropped beyond a threshold. We study how these two approaches affect overall service quality in the system. More specifically, the dynamic, tabula-rasa, approach considers reallocation of all tasks in every round given updated metrics from network. The partial, conservative, approach reconsiders only allocation of tasks whose operation is hindered by significant degradation of resources.

In order to study how drop rates are affected by neighbouring nodes and cross-traffic, we use the Castalia simulator [15]. Castalia provides an accurate radio and wireless channel model, modelling the error-prone behaviour of low-

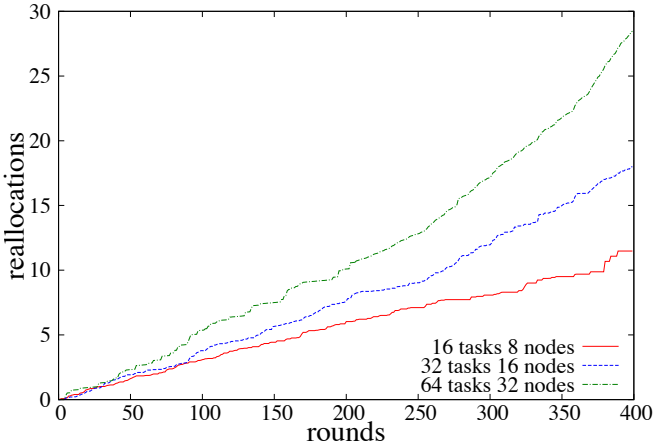


Figure 8. Cumulative task migrations in partial reallocation scheme

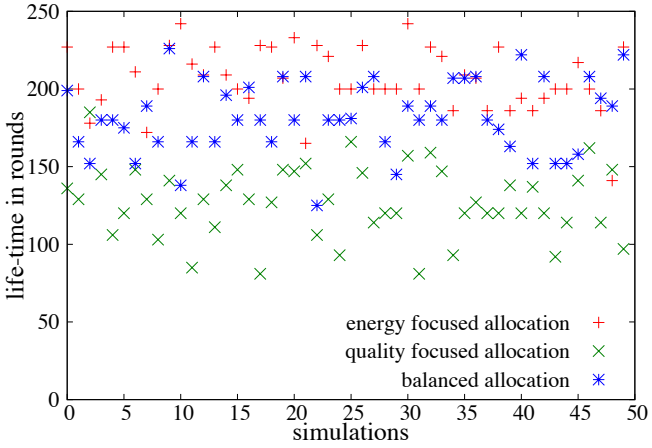


Figure 9. Application's lifetime based on task allocation scheme

power wireless links, which are common for wireless sensor networks. Node deployment in the network is random, but it retains a connected graph. We use a simple sensor quality classification model with three possible states: *accurate*, *degraded* and *faulty* with numerical score 3, 2 and 1 respectively. Each sensor may deteriorate at any round with a small probability. Finally, energy consumption is modelled in every node to increase proportional to the number of tasks allocated as this associates computational and communication costs.

We ran a set of 250 rounds comparing the two adaptive approaches and a static deployment that does not migrate tasks after the initial allocation. In the conservative approach, a task is migrated from a node if one of its required resources becomes *faulty*, the communication link's delivery rate with a dependent task drops below 50% or its node has disappeared from the network due to power depletion.

Figure 7(a) presents the average sensor data quality degradation of different approaches compared to the initial allocation over multiple simulation runs. The static, non-adaptive approach degrades fast being unable to take advantage of remaining sensors in the network with higher accuracy. The dynamic approach provides the least degradation, as it adapts immediately when a sensor degrades. Finally, partial reallocation follows a similar graceful degradation, but is slower to adapt to changes, being more conservative.

Similarly, figure 7(b) shows the average delivery rate of messages in the application in relation to the initial allocation. In all scenarios, the link quality drops from the initial allocation as a result of the traffic that the application introduces in the network. Message transmission affects drop rates in a node's neighbourhood. The static approach presents a sharp drop initially from which it is unable to recover and eventually degrades in time. Both adaptive approaches, while still affected by collisions in the medium, manage to maintain a stable trend on their delivery rates.

Finally, figure 8 shows the cumulative number of average task migrations in the case of partial reallocation for problems of different sizes. For reference, the tabula-rasa reallocation on every round, results in migration of almost every task, as it is very sensitive to subtle changes resulting in task thrashing between the network nodes. Partial reallocation significantly limits the amount of task movement inside the network making it a more viable alternative, while at the same time provides smooth service degradation on component failures. Even as the network size increases the number of migrations does not explode. An increase in the rate tasks are reallocated, however, can be noticed as the application grows older and nodes start running out of power.

C. Comparison with Energy Focused Deployment

Our model tries to find a trade-off between quality metrics and energy consumption. Consequently, we examine how network's lifetime is affected compared to a model that only considers task allocation based on nodes' residual energy. We test our approach to the objective function that is presented in [6]. We modify our original cost function pair in the objective equation 1 to only reflect the power consumed by computation and communication load imposed to a node by each task. Furthermore, we examine a third model where energy consumption is completely ignore in the utility functions and task assignment is solely based on the quality metrics.

For every model, we run 50 simulations with random task and network graphs of different sizes. For all simulations, we use the partial migration scheme. As expected, the balanced approach, followed for all previous simulations, falls in between the two extremes as shown in figure 9. The figure shows the running time in rounds of different simulation instances. Lifetime fluctuates in each model due to the random nature of graphs and manifesting faults. However, it is clearly observable that the energy focused allocation

supports longer running applications than the other two models. More specifically, the average running time for energy focused allocation is 206 rounds, 127 rounds for energy unaware allocation and 182 rounds for the balanced approach.

Consequently, the energy focused model provides 62% lifetime increase compared to an allocation scheme that ignores energy constraints. On the other hand, the balanced cost-function pair increases the average lifetime of the application, while it degrades more gracefully as components start to fail in the network. It sacrifices roughly 12% lifetime on average, compared to the pure energy aware model, while it improves quality metrics at the same time.

VII. Related Work

Task allocation has been studied extensively in traditional distributed systems, however, the setting differs significantly from WSNs. Such systems allocate tasks with different characteristics, usually involving batch processes and utilisation of parallelism to improve the system throughput, while nodes are not power constrained. In pervasive applications, allocation of the tasks is not motivated by load balancing and increased parallelism, but rather by distributing the monitoring process in an area to increase the observation range and provide redundancy for measured phenomena. Tasks are, typically, not computationally heavy and are event-driven rather than long batch processes that run once and terminate. Minimising communication is more crucial given that it is the most costly operation on power-constrained nodes.

In [4] the authors formulate the problem of task allocation in a similar way, as an ILP problem. However, the objective function in this case only accounts for minimising power consumption in nodes. The authors account for computation costs, selecting voltage power for CPU, and exchange of messages. They use LU factorisation and Fast Fourier Transform as applications to test the system. A similar approach is also presented in [5], however the authors make the assumption that only one task at a time is assigned on a node, which resembles batch processing rather than pervasive applications. Similarly in [16], single task allocation per node is considered, using utility and requirement functions for selection of nodes but also examining scenarios where new tasks are introduced during the network lifetime. In [17], [6] and [7], the authors define the problem in ILP, optimising the energy cost of the network, while the approaches in [8] and [18] also consider failures that are caused by battery depletion – the network tries to dynamically adapt when a node reaches low battery levels extending network lifetime.

Other approaches include a modified version of MinMin heuristic [19] for task allocation and agent-based approaches [20], [21]. In [20] a market model for nodes is built, where clients bid for a task based on their resource availability. Calculation of the bid is done locally by every node without global knowledge and the bids are evaluated centrally by the node that places the task on the market. The authors also propose an optimisation that reduces message exchange. Both paper also include the notion of deadlines for execution of tasks. In [21] task migration is motivated from application specifications, e.g. tasks that follow a moving target in the environment. The authors in [22] model sensors as agents that bid for tasks and a solution to the mission assignment is found by using GAP-E, a knapsack approximation algorithm.

In [23], the authors present a preliminary work on how the system can learn the quality of information in a WSN. The systems attempts to collect task satisfaction from competed tasks and employs an admission control mechanism based on available capacity of the network. Authors in [24] attempt to solve a similar problem by introducing a distributed algorithm, where nodes are collectively searching for a solution to the combinatorial problem of node configuration. Nodes diffuse good solutions to neighbours until the network stabilises. The approach, however, assumes each node has knowledge on other nodes' possible configurations. The expensive solution-search operation is taking place on power constrained nodes.

Most of the approaches above focus on long running tasks in WSNs similar to traditional distributed system and do not account for special issues of WSNs aside from the power constraints. Typical, monitoring application and data aggregation processes, which are the most prominent applications of pervasive systems, do not fit well to these properties. We argue that the approach proposed in this paper is closer to their requirements and accounts for quality aspects of pervasive applications, such as packet delivery rate and quality of sensor readings.

Furthermore, none of these approaches has attempted to extract missions graphs and constraints directly from mission specification. The simplicity of specification using policies, in the Starfish framework, allows for automating this process, while being expressive enough for configuring network applications.

VIII. Conclusions

We have presented in this paper a framework for autonomic task allocation in complex pervasive systems that consist of several components. We have presented the formal specification of the problem in ILP, an NP-hard problem, as well as an approximation method using simulated annealing, a general meta-heuristic method, that searches a broad spectrum of the problem space. Future work will consider that some critical tasks are more important than others and should be given priority with respect to migration or to maximise lifetime.

We have further demonstrated the integration with Starfish framework, an instantiation of the Self-Managed Cell architecture for embedded sensor devices. SMC constructs, such as roles, missions and policies, allow an easy translation of

an application written in the Finger2 policy system into a task graph that is the input of the allocation algorithm.

ACKNOWLEDGMENT

We would like to thank the National Institute of Informatics, Japan for supporting Themistoklis Bourdenas visit to the Institute and acknowledge support from EPSRC for Grant EP/EO251881 AEDUS2: Adaptable Environments for Distributed Ubiquitous Systems.

REFERENCES

[1] I. Akyildiz, T. Melodia, and K. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921–960, 2007.

[2] J. Kephart and D. Chess, "The vision of autonomic computing," *IEEE Computer*, pp. 41–50, Dec 2003.

[3] M. Huebscher and J. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, p. 7, 2008.

[4] S. Abdelhak, C. Gurram, S. Ghosh, and M. Bayoumi, "Energy-balancing task allocation on wireless sensor networks for extending the lifetime," *53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 781–784, 2010.

[5] Y. Tian, E. Ekici, and F. Ozguner, "Energy-constrained task mapping and scheduling in wireless sensor networks," *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005*, pp. 8 pp.–218, 2005.

[6] A. Pathak and V. Prasanna, "Energy-efficient task mapping for data-driven sensor network macroprogramming," *IEEE Transactions on Computers*, vol. 59, no. 7, pp. 955–968, 2010.

[7] P. Aghera, D. Krishnaswamy, D. Fang, A. Coskun, and T. Rosing, "Dynaheal: Dynamic energy efficient task assignment for wireless healthcare systems," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1661–1664, 2010.

[8] Y. Chen, W. Guo, and G. Chen, "A multi-agent-based adaptive task allocation algorithm in wireless sensor networks," *International Conference on Information Engineering and Computer Science,*, pp. 1–4, 2009.

[9] T. Bourdenas, M. Sloman, and E. Lupu, "Self-healing for pervasive computing systems," *Architecting Dependable Systems VII*, pp. 1–25, 2010.

[10] T. Bourdenas and M. Sloman, "Starfish: policy driven self-management in wireless sensor networks," *Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2010.

[11] K. Twidle, E. Lupu, M. Sloman, and N. Dulay, "Ponder2: A policy system for autonomous pervasive environments," in *ICAS*. IEEE, 2009.

[12] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*. New York, NY, USA: ACM, 2004, pp. 81–94.

[13] W. Munawar, M. Alizai, O. Landsiedel, and K. Wehrle, "Dynamic tinyos: Modular and transparent incremental code-updates for sensor networks," *IEEE International Conference on Communications (ICC)*, pp. 1 – 6, 2010.

[14] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.

[15] H. Pham, D. Pediaditakis, and A. Boulis, "From simulation to real deployments in wsn and back," *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007*, pp. 1–6, 2007.

[16] M. Johnson, H. Rowaihy, D. Pizzocaro, A. Bar-Noy, S. Chalmers, T. L. Porta, and A. Preece, "Sensor-mission assignment in constrained environments," *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[17] H. Park and M. Srivastava, "Energy-efficient task assignment framework for wireless sensor networks," *Center for Embedded Networking Sensing (CENS), Technical Reports*, 2003.

[18] J. Zhu, J. Li, and H. Gao, "Tasks allocation for real-time applications in heterogeneous sensor networks for energy minimization," *Eighth International Conference Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, vol. 2, pp. 20–25, 2007.

[19] W. Xiao, S. Low, C. Tham, and S. Das, "Prediction based energy-efficient task allocation for delay-constrained wireless sensor networks," *SECON Workshops' 09. 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009*, pp. 1–3, 2009.

[20] N. Edalat, W. Xiao, C.-K. Tham, E. Keikha, and L.-L. Ong, "A price-based adaptive task allocation for wireless sensor network," *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pp. 888–893, 2009.

[21] S. Suenaga, N. Yoshioka, and S. Honiden, "Group migration by mobile agents in wireless sensor networks," *The Computer Journal*, 2009.

[22] T. Le, T. Norman, and W. Vasconcelos, "Agent-based sensor-mission assignment for tasks sharing assets," *Proc. Third Int. Workshop on Agent Technology for Sensor Networks (ATSN)*, pp. 33–40, 2009.

[23] C. Liu, C. Bisdikian, J. Branch, and K. Leung, "Qoi-aware wireless sensor network management for dynamic multi-task operations," *7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 1–9, 2010.

[24] N. Salazar, J. Rodriguez-Aguilar, and J. Arcos, "Self-configuring sensors for uncharted environments," *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 134 – 143, 2010.