Australian Journal of Basic and Applied Sciences, 4(8): 3455-3466, 2010
ISSN 1991-8178

Optimal Real-Time Task Allocation in Wireless Sensor Actor Networks

[1]Mohsen Sharifi, [1]Hossein Momeni and [2]Vahid Rafe

[1,2]School of Computer Engineering
[1]Iran University of Science and Technology
[2]Arak University, Arak, Iran

Abstract: Real-time communication and coordination account as one of the most important challenges in wireless sensor actor networks (WSANs). However, existing mechanisms and architectures for WSANs suffer from the lack of any optimal real-time task allocation to support real-time communications. In this paper we present a formal structure for optimal task allocation in WSANs. To do so, periodic tasks are determined, and then appropriate algorithms are designed for sensing tasks and acting tasks. To analyze the correctness of these algorithms, we propose a formal model for WSANs using graph transformation systems. We show that the proposed algorithms guarantee that the tasks complete their activities before their deadlines expire. Simulation results showed an improvement of 65 percent in deadline hit ratio comparing our algorithms to FIFO algorithm.

Key words: Wireless Sensor Actor Network, Real-Time, Task Allocation, Sensing Tasks, Acting Tasks, Graph Transformation Systems, Formal Analysis

INTRODUCTION

Wireless sensor actor networks (WSANs) consist of a group of sensor and actor nodes that perform distributed sensing and acting tasks (Xia, 2008). Sensors are able to sense environmental information and actors are able to act upon environment. These nodes communicate wirelessly and dynamically to monitor and control environment. Fig. 1 shows a typical architecture of WSANs (Akyildiz *et al*., 2004). In this architecture, the task manager node sends its request through Internet and satellite to the sink and the sink as a central computer dispatches these requests to designated sensor and actor nodes that in turn send back the results to the sink. WSANs have been used for a variety of applications, such as surveillance systems and fire detection (Xia *et al*., 2007).

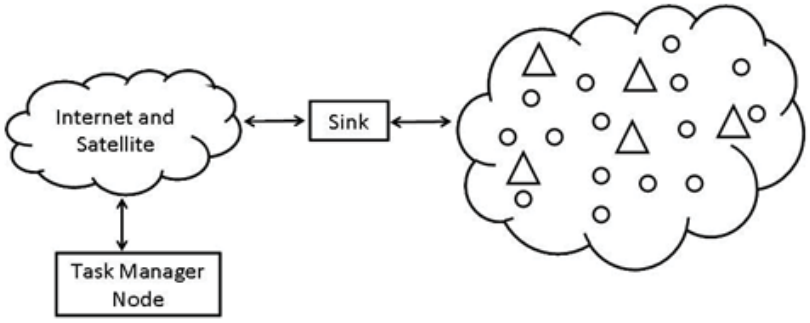

Fig. 1: A typical architecture of WSANs

The existence of actors in an environment is demonstrative of time in performing some tasks; hence real-timeliness is an important issue in WSANs to guarantee that tasks of an application are distributed based on limited computation and communication resources in the network in such a way to meet the deadlines.

In spite of the importance of real-time requirements in WSANs, this issue had not been totally resolved yet. Existing researches, e.g., (Shah *et al*., 2006), have tried to minimize delays in packet transmission, in real-time routing and in coordination, but fall short of guaranteeing task completion time before specified

Corresponding Author: Mohsen Sharifi, School of Computer Engineering, Iran University of Science and Technology

deadlines expire. To provide such guarantees, tasks must be allocated to network nodes considering the deadlines. This had only been pursued in wireless sensor networks (WSNs).

Park *et al*. (2003) have presented an energy-efficient task allocation framework for WSNs. Their proposed framework uses graph descriptions to decompose tasks and then allocate them to appropriate sensor nodes. The allocation is made in a way to minimize a cost function representing energy consumption.

Younis *et al*. (2003) have presented a task allocation algorithm for gateways (cluster heads) in sensor networks. The lifetime of network is maximized by this algorithm. To simplify the scheduling of tasks on gateways, they have assumed that the time for processing of collected data lags by at least one cycle. An optimization scheme for task allocation to gateways is presented. The objective of this optimization is to maximize the lifetime of all gateways.

Yu *et al*. (2005) have proposed an energy-balanced allocation of a real-time application onto a single-hop cluster of homogeneous sensor nodes connected by multiple wireless channels. The proposed task allocation algorithm allocates the tasks onto sensor nodes, performs the voltage settings of tasks, allocates the communication activities onto channels, and finally schedules the computation and communication activities. Tian *et al*. (2005) have presented an energy-constrained task mapping and scheduling mechanism called EcoMapS that maps and schedules the tasks of an application with minimum schedule length subject to energy consumption constraints.

Given this background in WSNs, in this paper we expand the line of work presented in Momeni *et al*. (2009) to propose an optimal task allocation mechanism for WSANs by allocating end-to-end tasks to sensor nodes and actor nodes considering the real time deadlines of tasks. Our mechanism consists of two algorithms, one for sensing end-to-end tasks, and another one for acting end-to-end tasks (Rafe et. al., 2009). Candidate sensor nodes and actor nodes that have enough energy and can perform the tasks are identified first, and then tasks are dispatched to them based on deadlines.

To do so, we first model formally the required features of WSANs by graph transformation systems. Graph transformation has recently become more and more popular as a general formal modeling language (Baresi *et al*., 2008). Moreover, it is a natural formalism for modeling systems that have graph representations. These motivated us to choose graph transformation for modeling WSANs. We use this model to analyze the soundness of our algorithms. Using the AGG toolset, we then analyze the modeled WSANs and check the results to assure the correctness of the model (Momeni *et al*. 2009). Finally, the efficiency of our approach is evaluated by simulation using VisualSense (Baldwin *et al*., 2004).

Assumptions and Terminologies:

In this section we state our assumptions for task allocation in WSANs, besides a brief description of graph transformation systems.

Task Allocation in WSANs:

We have made the following assumptions for task allocation in WSANs:

- The tasks acting on sensed data are considered periodic, because most sensory processing is periodic.
- The transfer of data from every sensor to the sink is considered as an end-to-end task.
- All tasks are considered non-preemptive, i.e. their execution goes to completion without interruption.
- The cluster architecture for sensor nodes is assumed to cater for scalability and management of sensors.
- Acting Tasks are performed by just one actor. These tasks are named single actor tasks (SATs).
- A semi-automated architecture (Akyildiz *et al*., 2004) is assumed for sending data to the sink (Fig. 2).

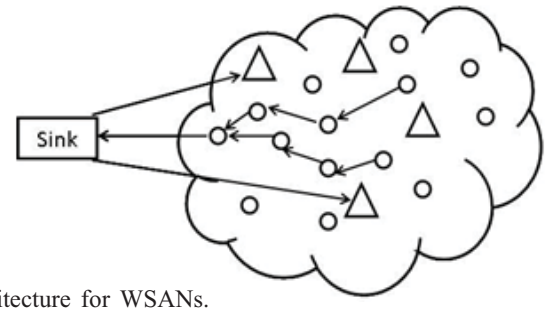

Fig. 2: A semi-automated architecture for WSANs.

Apart from the noted assumptions, we have used a number of terminologies that are described hereafter. The commonest type of tasks in real-time (monitoring) systems are the end-to-end tasks wherein the first job in each task is periodic but the remaining jobs in the task are sporadic. Formally, we define an end-to-end task as follows:

$$T_i =< \delta_i, period_i, deadline_i >, 1 \le i \le N \quad (1)$$

wherein represents an infinite sequence of jobs:

$$\delta_i =< j_1, j_2, ..., j_k >, 1 \le i \le k \quad (2)$$

Based on this tuple, the execution of j_i starts only after j_{i-1} has completed its execution, where represents the period of T_i meaning that T_i is executed once every $period_i$ units of time. Furthermore, the period of j_i is equal to $period_i$, but other jobs are sporadic. The deadline for T_i is equal to $deadline_i$, i.e. the time that execution of T_i must be completed.

The execution of the first job in δ_i starts with the period of the task, while the executions of other jobs in the task start based on the job scheduling policy. We have chosen a greedy policy for scheduling these jobs, wherein the execution of each job from an end-to-end task starts after the completion of its previous job.

We differentiate two types of end-to-end tasks: sensing and acting. These tasks include some jobs and are formalized as follows:

- A sensing task is represented as follows:

$$\tau_i =< sense_i, SendSNtoCH_i, SendCHtoSink_i > \quad (3)$$

where the first job, i.e. $sense_i$, sends a request for sensing from the sink to cluster heads in WSANs. The second job, i.e. $SendSNtoCH_i$, sends gathered information of sensor nodes to its cluster head. Finally, the last job, i.e. $SendCHtoSink_i$, sends this information to the sink.

- An acting task is represented as follows:

$$\varphi_i =< Action_i, SendACtoSink_i > \quad (4)$$

where $Action_i$ is a job that actors can perform. The sink uses the tuple space to get information about actors and sends $Action_i$ to an appropriate actor. After execution of this job, the actor sends the returned results to the sink based on $SendACtoSink_i$

For remote task communications we use a tuple space communication architecture originally proposed in Linda (Gelernter, 1985). Tuples are collections of passive data values. A tuple space is a pool of shared information, where tuples can be inserted, removed or read (Kuorilehto *et al*., 2005).

We introduce two tuple spaces that share information about sensors and actors among cluster heads and the sink, respectively. Cluster heads and the sink can use the tuple space to retrieve information about the status of sensor nodes such as about the remaining energy of each node. Fig. 3 shows tuple space architecture for WSANs.

We define two tuple spaces as follows:

- If $S = \{SN_1, SN_2, ..., SN_L\}$ is a set of L sensor nodes and $Sense = \{Sense_1, Sense_2, ..., Sense_M\}$ is a set of M sensing jobs, then the first tuple space is defined as follows:

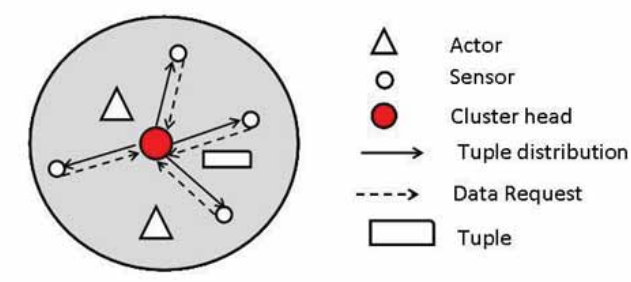

Fig. 3: A tuple space architecture for WSANs

$$\Delta_{j,k} =< sense_{j,k}, delay_{j,k}, power_{j,k} > \tag{5}$$

where $1 \le k \le M, 1 \le j \le l$ and $\Delta_{j,k}$ denotes that the jth sensor performs the kth sensing job with dilation time $delay_{j,k}$ and power consumption $power_{j,k}$.

- If $A = \{A_1, A_2, ..., A_Z\}$ is a set of Z actor nodes and $Action = \{Action_1, Action_2, ..., Action_X\}$ is a set of X acting jobs, then the second tuple space is denoted by:

$$\psi_{j,k} =< Action_{j,k}, delay_{j,k}, power_{j,k} > \tag{6}$$

where $1 \le k \le X$ and $\psi_{j,k}$ denotes that the jth actor performs the kth acting job with dilation time $delay_{j,k}$ and power consumption $power_{j,k}$.

Graph Transformation Systems:

The mathematical foundation of graph transformation systems goes back to the year 1970 in reaction to shortcomings in the expressiveness of classical approaches to rewriting (e.g. Chomsky grammars) to deal with non-linear grammars. Here, we describe graph transformation briefly, as a mean for modeling. For more information about theoretical backgrounds and semantics of graph transformation, interested readers are referred to Baresi *et al*. (2002) and Ehrig *et al*. (1999).

Definition 1 (Attributed Type Graph Transformation):

An attributed type graph transformation system is a triple AGT=(TG,HG,R), where TG is the type graph, HG is the host graph and R is the set of rules.

Definition 2 (Type Graph):

Let TG_N be a set of node types and TG_E be a set of edge types. Then a type graph TG is a tuple: $TG=(TG_N,TG_E,src,trg)$, with two functions src: $TG_E \rightarrow TG_N$ and trg: $TG_E \rightarrow TG_N$ that assigns to each edge a source and a target node. Each node type NT in TG_N is a triple: NT=(Mult, Attr,O), where Mult is the multiplicity of the node and it is a pair: Mult=(min,max), where min≤max and the pair shows the minimum and maximum nodes of type NT in the host graphs. Attr is the set of its attributes and it is a tuple: Attr=(SN,V,X,sort), where SN is a set of sort names (or type names), V is a set of attribute values including a subset $X \subseteq V$ of variable names, and a function sort: $V \rightarrow SN$ associating every value and variable with a sort (type). O is the set of its outgoing edges (associations) with corresponding multiplicity and destination node. More precisely, each outgoing edge OE in O is a pair: OE=(Card,Dest), where Card is formally defined by two pairs of functions minSrc,minTrg: $TG_E \rightarrow N^{\cup}\{0\}$ and maxSrc,maxTrg: $TG_E \rightarrow N^{\cup}\{*\}$ with minSrc(OE)≤maxSrc(OE) and minTrg(OE)?maxTrg(OE), and Dest is the destination node of the edge.

Definition 3 (Host Graph):

A host graph HG, also called instance graph over TG, is a graph equipped with a graph morphism typeG: HG→TG that assigns a type to every node and edge in HG.

Definition 4 (Graph Rules):

In this paper, we have followed the algebraic double pushout approach (DPO) to graph transformation as first introduced by Ehrig *et al*. for untyped graphs in (Ehrig *et al*, 1999). A graph transformation rule P over an attributed

type graph TG is given by $P = (L \xleftarrow{l} K \xrightarrow{r} R, type, NAC)$, where:

- $L \xleftarrow{l} K \xrightarrow{r} R$ is a rule span with injective graph morphisms l,r and graphs L (Left Hand Side or LHS), K (gluing graph) and R (Right Hand Side or RHS) typed over TG.
- *type=(typeL: L→TG, typeK: K→TG, typeR: R→TG)* is a triple of morphisms, and
- NAC is a set of triples nac=(N,n,typeN) with N being a graph, n: L→N a graph morphism, and typeN: N→TG a morphism.

The application of a rule to a host graph H, replaces a matching of the LHS in H by an image of the RHS. This is performed by (1) finding a matching of LHS in H, (2) checking the negative application condition NAC that prohibits the presence of certain nodes and edges, (3) removing a part of the host graph that can be mapped to LHS but not to RHS, yielding the context model, and (4) gluing the context model with an image of the RHS together by adding new nodes and edges that can be mapped to the RHS but not to the LHS, and obtaining the derived model H.

System Model:

In this section, we describe our approach to model the WSANs using graph transformation systems. As it was mentioned, in a graph transformation system, the type graph shows the metamodel of the system. Hence, designing the type graph is the first step to model the system.

Fig. 4 shows our proposed type graph as the metamodel of the system. It comprises of 10 nodes with different types. For example, node "clock" shows the current time in the model. It has an attribute named "Time" to show the current time. We use this node to simulate the time. The behaviors that must pertain (i.e. the rules that must be applied) at the same time on the model do not change the time but other rules change it. The node "Sensor" is an abstract node from which the two nodes "ClusterHead" and "TypicalSensor" have inherited. In fact, each sensor in the model is either a cluster head or a typical sensor.

Also, there is an association edge between "ClusterHead" and "TupleSpace". This means that each cluster head in the model has a tuple space that contains triples called "Tuple". Each "TypicalSensor" stores its current status as a "Tuple" in the "TupleSpace". We have designed this type graph based on the assumptions in Subsection 2.1. For example, the node "Tuple" contains all the attributes we defined in the previous section. The type graph in Fig. 4 shows our proposed syntax for the models. According to this type graph, we can model the initial configuration of the WSAN as a host graph. For example, Fig. 5 shows a small WSAN with 3 sensors, 1 cluster head, 1 sink, 1 tuple space and so forth. In general, the host graph is served as the starting point for simulation and analysis.

We define graph rules to implement the desired semantics (based on the proposed algorithms described in the next section). For each behavior of the model, one or more graph rule(s) is defined. For example, consider the rule in Fig. 6 that shows the creation of a sensing task. The left hand side of this rule says if there is a sink in the model whose "SensingDone" attribute is False and there is not a "SensingTask" associated with it (negative application condition), then a sensing task is generated by application of this rule. For this rule, we have assumed that the kind of the task is temperature sensing and its deadline is 4. For other cases (i.e., different kinds of sensing tasks and different deadlines) we can define similar rules.

As another example, consider the rule in Fig. 7 that shows the process of assigning a sensing task to cluster heads by the sink node. The LHS and NACs together imply that if the sink has a task and this task has not been assigned to a cluster head, when both nodes have the same kind, then the task must be assigned to the cluster head. After assigning the task to all cluster heads, which have the same kind as the task, then the link between the sink node and the SensingTask node is omitted by another rule that is not shown here.

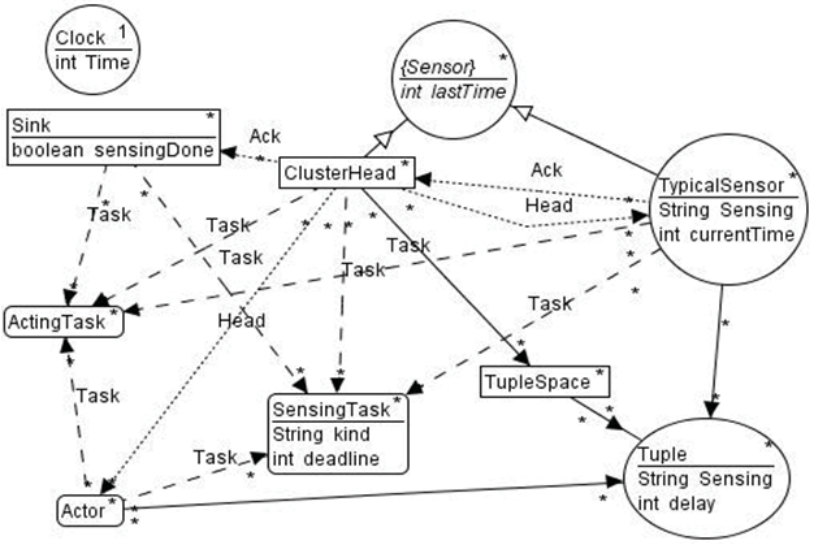

Fig. 4: The proposed type graph for WSANs model

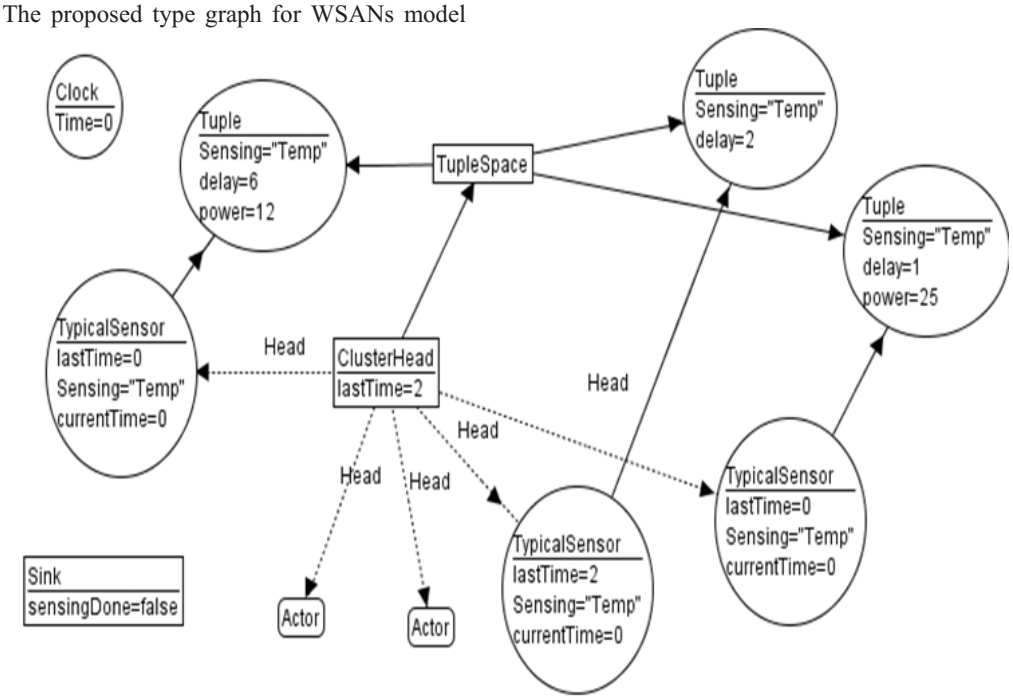

Fig. 5: A host graph showing the initial configuration of a small WSAN

Based on different scenarios in our proposed algorithms, we can define all necessary rules to support them. As exemplars, some of these rules such as 'create sensing task', 'create acting task' and alike are shown in Fig. 8 and Fig. 9. Using this model we can simulate and analyze the correctness of the algorithms. The algorithms themselves are presented in the next section

Proposed Algorithms:

Two algorithms are proposed for real-time task allocation. The first algorithm (*Algorithm1*) is distributed and is run by cluster heads. According to this algorithm, cluster heads decide if periodic tasks that come from the sink are allocated to appropriate sensors or not. If a cluster head finds that there is a sensor node that is

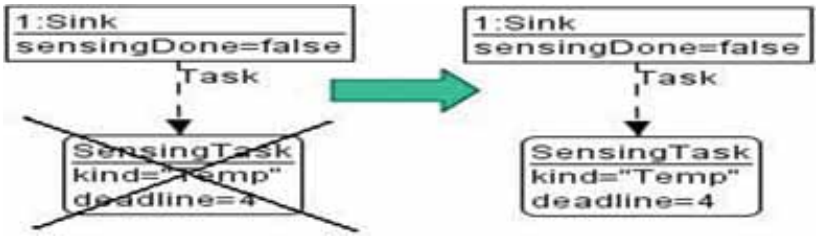

Fig. 6: The rule for creating a sensing task

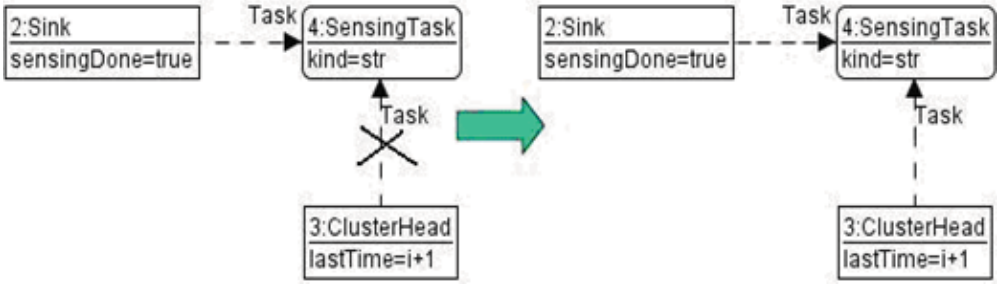

Fig. 7: The rule for assigning a sensing task

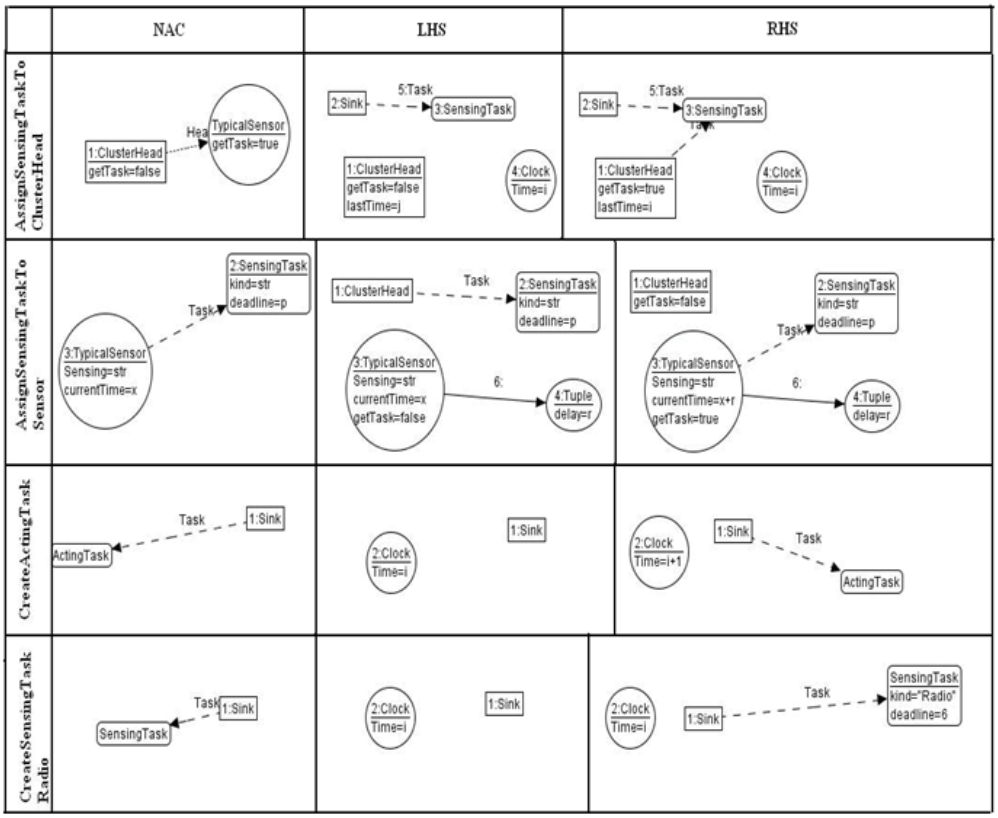

Fig. 8: Some of the proposed graph rules

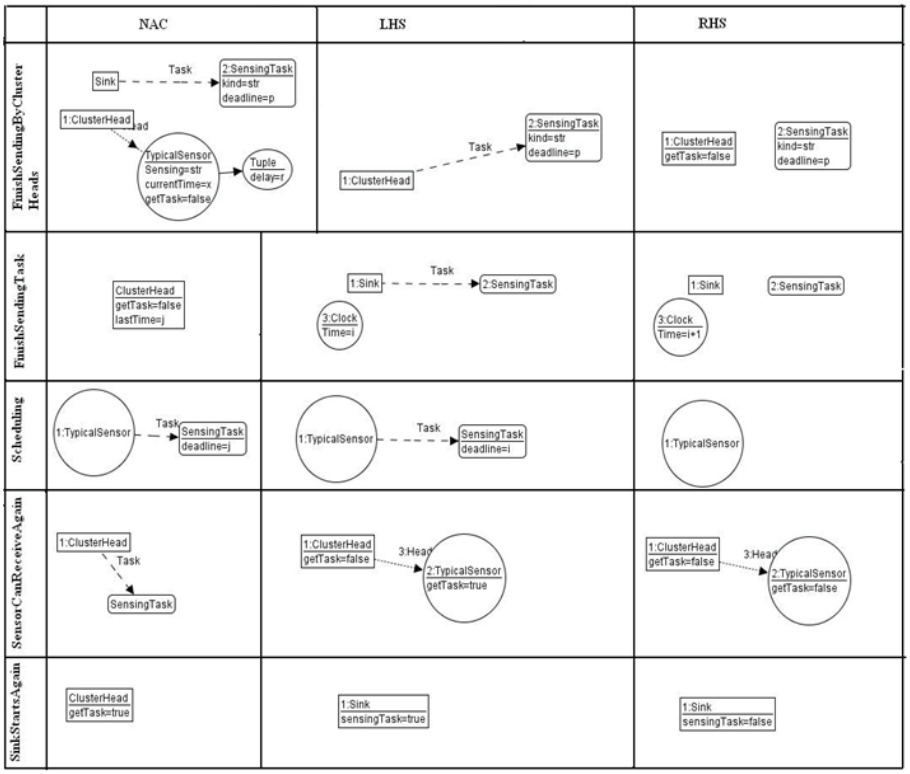

Fig. 9: Other proposed graph rules

able to perform a requested sensing job, the cluster head allocates that job to that node; otherwise the cluster head rejects that job and informs the sink about the rejection.

Algorithm1 details the operations that are run by each cluster head. These operations were modeled by different graph transformation rules presented in the previous section.

Algorithm1 is designed to allocate periodic sensing tasks to sensors in WSANs and is executed at the cluster head level; cluster head waits to receive sensing tasks from the sink. Each cluster uses the tuple space Δ to determine if any sensor can sense the sensing task or not. If any sensor in the cluster can meet the deadline of the task, then the next important condition that should be satisfied is that "*the deadline of the end-to-end task should be less than delays of jobs"*:

$$deadline_i = delay_{j,k} + SendSNtoCH_i.delay + SendCHtoSink_i.delay \quad (7)$$

***Algorithm1*. Sensing Task Allocation**

1: cluster head awaits the reception of a task τ_i from the sink.

2: find all $\Delta_{j,k}$ for each $j, k\ (1 \le k \le M, 1 \le j \le l)$ such that $sense_i$ in τ_i is equal to $sense_{j,k}$

3: if $\Delta_{j,k}$ not found in step 2, do not accept τ_i and go to step 1

4: for all $\Delta_{j,k}$ found in step 2 above if $deadline_i = delay_{j,k} + SendSNtoCH_i.delay + SendCHtoSink_i.delay$

then do not accept τ_i and go to step 1

5: send τ_i to a selected sensor in the cluster

6: if all sensors in the cluster do not accept τ_i , then go to step 1

7: allocate τ_i to the sensor and go to step 1

After the sink receives environmental information from the cluster head, it detects the event region based on this information and sends an appropriate acting task to the actors residing in that region. *Algorithm2* presents the details.

The acting task allocation described in *Algorithm2* is initiated when the sink receives environmental information from cluster heads. According to this information, the acting task φ_i is performed. The first job of this task is $action_i$ that actors can perform. In line 2 of this algorithm, the sink uses the Ψ tuple space to retrieve information about the status of actors, such as the type of acting job that actors can perform. $Action_{j,k}$ denotes that actor *j* can perform action *k*, so if $action_i$ (the first job in φ_i) is equal to $Action_{j,k}$, then the sink can assign $action_i$ to the *j* actor.

***Algorithm2*. Acting Task Allocation**

1: sink waits to receive results from cluster heads
2: find all $\psi_{j,k}$ for each j, k ($1 \le k \le X, 1 \le j \le Z$) such that the first job in φ_i , $action_i$ is equal to $Action_{j,k}$
3: if $\psi_{j,k}$ is not found in step 2, do not accept φ_i and go to step1
4: for all $\psi_{j,k}$ that is found in step 2 if $deadline_i > delay_k + SendACtoSink_i.delay$
then do not accept φ_i and go to step 1
5: send φ_i to the selected actor
7: allocate φ_i to the actor and go to step 1

But before the assignment of $action_i$, the second condition should be satisfied. This condition checks if the selected actor meets the deadline o f task, the sum of $delay_k$ (the delay of performing job *k*) and $SendACtoSink_i.delay$ (the delay of sending action result to the sink) should be less than the deadline:

$$deadline_i > delay_k + SendACtoSink_i.delay \quad (8)$$

The first actor that replies to the request and satisfies the above conditions performs the action and sends a signal to the sink.

These two allocation mechanisms guarantee that sensor and actor nodes just accept tasks that can finish executing them before their deadlines expire.

Verification and Validation:

This section describes two approaches: formal model checking and simulation to verify and validate our proposed models and algorithms, respectively.

Model Checking:

To verify our proposed model formally, we have used the approach presented in Rafe *et al*. (2009), wherein graph transformation specifications are verified through a model checker called Bogor (Robby *et al*., 2003). In fact, the authors present an approach to encode graph transformation systems to the BIR -the input language of Bogor- for verification. The main steps of the proposed approach to encode graph transformation systems are summarized as follows.

- The first step is the encoding of type graph into BIR to define the data structures needed later. Each node type is translated into a 'record' containing all the information in the node (including the attributes and associations). Another record is added too, which represents the graph itself.
- The second step is the instantiation of the host graph. To do so, a variable of type 'graph' must be declared in BIR.

- In the third step, the rules must be rendered into BIR.
- In the end, the properties must be translated into BIR. To do so, we mimic GROOVE (Rensink, 2004) and CheckVML (Schmidt, 2003) to declare the properties. We use special graph rules in which LHS shows the positive conditions, NAC shows the negative ones and RHS is similar to LHS while it does not change the states.

We can now use this approach to define some important properties and then verify them. For example, we have considered the following properties of the proposed model:

- If an actor accepts a job, then eventually it must finish the job before its deadline; this is a liveness property.
- Sinks cannot generate any more tasks while their generated tasks have not been consumed; this is a safety property.
- The sequence of distributing tasks, performing etc. must infinitely continue; this is a deadlock-free property.
- There must exist an state in which all actors have an assigned job; this is a reachability property.

After testing all the above properties, we can be sure about the soundness of the model.

Simulation:

The effectiveness of our approach is validated by simulating our proposed algorithms using the VisualSense simulator (Baldwin, 2004). In our simulations, we created a WSAN consisting of varying numbers of sensors and actors, wherein nodes were randomly placed in an area of 800m×600m. We considered 4 clusters with 4 sensors and 1 actor in each cluster. Each sensor node was assumed to have an initial energy of 360 Joules. The network was considered non-functional if its overall energy level reached less than 50 Joules. The radio range and sensing range for a sensor node was assumed to be 200m and 20m, respectively. The radio range and acting range for an actor node was assumed to be 200m and 40m, respectively

We first modeled the sink node, sensor nodes and actor nodes that contain TypedCompositeActor and include time component, send message component and receive message component. Each TypedCompositeActor has a TypedAtomicActor that acts as the processor of each node.

In the experiments, we counted the number of tasks that were rejected by nodes and the number of deadlines that were missed. We compared our approach with FIFO algorithm and observed that our approach provides the least rejected tasks and lower number of deadline misses (Fig. 10).

Conclusion:

Wireless sensor actor networks (WSANs) have attracted considerable attention and usage over the past few years especially in applications running in hostile and remote areas such as military and combat field surveillance and environment monitoring applications. These applications often have real-time requirements and their tasks must be performed by sensor and actor nodes within deadline. This issue had not been researched in WSANs. So in this paper we proposed a two-level allocation mechanism and introduced two algorithms to prevent missing the deadlines of such tasks. For this purpose, before allocating tasks to nodes, we checked the

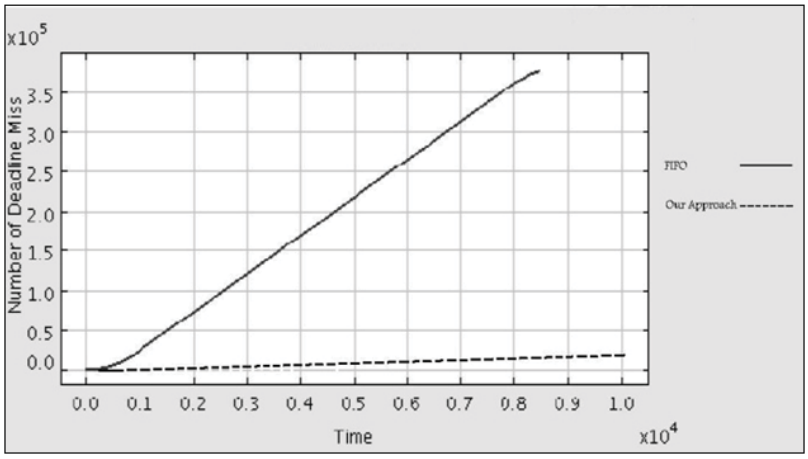

Fig. 10: Deadline misses

feasibility of execution of tasks on those nodes based on their remaining energy and deadline requirements. We first modeled formally the required features of WSANs by graph transformation systems. We then used this model to analyze the soundness of our algorithms. We analyzed the modeled WSANs and checked the results to make sure the model is correct. Finally, the efficiency of our approach was evaluated by simulation, showing an improvement of 65 percent in deadline hit ratio compared to FIFO algorithm.

We are currently extending our approach to automated architecture of WSANs wherein sensory data are reported directly to actors rather than through the sink, actors coordinate and decide to perform appropriate actions directly without the intervention of the sink.

REFERENCES

Akyildiz, I.F., I.H. Kasimoglu, 2004. Wireless Sensor and Actor Networks: Research Challenges. Ad Hoc Networks, 2: 351-367.

Baldwin, P., S. Kohli, E.A. Lee, X. Liu, Y. Zhao, C. Brooks, N.V. Krishnan, S. Neuendorffer, C. Zhong and R. Zhou, Visualsense, 2004. Visual Modeling for Wireless and Sensor Network Systems. Electronics Research Laboratory, College of Engineering, University of California.

Baresi, L., V. Rafe, A.T. Rahmani and P. Spoletini, 2008. An Efficient Solution for Model Checking Graph Transformation Systems. Electronic Notes in Theoretical Computer Science (ENTCS), 213: 3-21.

Baresi, L.R. Heckel, 2002. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. First International Conference on Graph Transformation, LNCS, 2505: 402-429.

Ehrig, H., G. Engels, H. Kreowski, G. Rozenberg, 1999. Handbook on Graph Grammars and Computing by Graph Transformation, Applications. Languages and Tools, Published by, World Scientific.

Gelernter, D., 1985. Generative Communication in Linda. ACM Transaction, Programming Languages and Systems, 7(1): 80-112.

Kuorilehto, M., M. Hännikäinen, T.D. Hämäläinen, 2005. A Survey of Application Distribution in Wireless Sensor Networks. EURASIP Journal on Wireless Communications and Networking, 5(5): 774-788.

Momeni, H., M. Sharifi and Seed Sedighian, 2009. A New Approach to Task Allocation in Wireless Sensor Actor Networks. Computational Intelligence, Communication Systems and Networks Conference (CICSYN 2009), Indore, India.

Momeni, H. V. Rafe, M. Sharifi, A.T. Rahmani, 2009. A Graph Transformation-based Approach to Task Allocation in Wireless Sensor Actor Networks. Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, USA, 1-3 July: 609-612.

Park, H., M.B. Srivastava, 2003. Energy-Efficient Task Assignment Framework for Wireless Sensor Networks. Center for Embedded Networking Sensing (CENS), Technical Report.

Rafe, V., A.T. Rahmani, L. Baresi, P. Spoletini, 2009. Towards Automated Verification of Layered Graph Transformation Specifications. Journal of IET Software, 3(4): 276-291.

Rafe, V., H. Momeni, M. Sharifi, 2009. Energy-Aware Task Allocation in Wireless Sensor Actor Networks. Second International Conference on Computer and Electrical Engineering (ICEEE 2009), Dubai, UAE.

Robby, E., M. Dwyer, J. Hatcliff, 2003. Bogor: An Extensible and Highly-Modular Software Model Checking Framework. Proceedings of the 9th European Software Engineering Conference, 267-276.

Rensink, A. 2004. The GROOVE Simulator: A Tool for State Space Generation. In Applications of Graph Transformations with Industrial Relevance (AGTIVE), Lecture Notes in Computer Science, 3062: 479-485.

Schmidt, A., D. Varró, V.M.L. Check, 2003. A Tool for Model Checking Visual Modeling Languages. Proceedings of 6th International Conference on the Unified Modeling Language (UML), Lecture Notes in Computer Science, 2863: 92-95.

Shah, G.A., M. Bozyiğit, Ö.B. Akan and B. Baykal, 2006. Real-Time Coordination and Routing in Wireless Sensor and Actor Networks. Lecture Notes in Computer Science (LNCS), 4003: 365-383.

Tian, Y., E. Ekici, F. Ozguner, 2005. Energy-Constrained Task Mapping and Scheduling in Wireless Sensor Networks. Proceedings of IEEE International Conference on Mobile Ad-hoc and Sensor Systems, 8-16.

Xia, F., 2008. QoS Challenges and Opportunities in Wireless Sensor/Actuator Networks. Sensors Journal, 8(2): 1099-1110.

Xia, F., Y.C. Tian, Y. Li, Y. Sun, 2007. Wireless Sensor Actuator Network Design for Mobile Control Applications. Sensors Journal, 7: 2157-2173.

Younis, M., K. Akkaya, A. Kunjithapatham, 2003. Optimization of Task Allocation in a Cluster-Based Sensor Network. Proceedings of 8th IEEE International Symposium on Computers and Communication, 329-334.

Yu, Y., V.K. Prasanna, 2005. Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks. Mobile Networks and Applications, 10: 115-131.