ADAPTATION IN SENSOR-ACTUATOR NETWORKS: A CASE STUDY

Mahesh Arumugam, Sandeep S. Kulkarni and Karun N. Biyani
Department of Computer Science and Engineering
Michigan State University, East Lansing MI 48824

***Abstract*— In this paper, we concentrate on component adaptation in sensor-actuator networks. Component adaptation is the process of dynamically replacing components in an application in response to changes in the environment conditions. Since the normal operation of a sensor-actuator network is often hindered by faults (e.g., node failures, message losses, synchronization errors), it is important that the application adapts to the given conditions by dynamically composing suitable components. On the other hand, the applications are time-critical in nature and, hence, it is necessary to ensure that the network provides the required functionalities during adaptation. In this paper, we propose an approach for component adaptation in sensor-actuator networks by systematically using the distributed reset framework that provides assurance guarantees during adaptation. And, we perform a case study on a vibration control application.**

Keywords: adaptation, distributed reset, sensor-actuator networks, vibration control experiment

I. INTRODUCTION

In the recent years, sensor-actuator networks (SANETs) are becoming popular in distributed control of physical phenomenon and in active tracking of undesirable objects. For example, a SANET is useful in reducing the vibroacoustic perturbations of fairing structures during the liftoff and the ascent to orbit of satellite launches [1]. Such networks reduce the structural weight and improve the reliability.

One challenge in SANETs is that the nodes are resource constrained. Hence, they need to collaborate among each other to perform a certain task. Such collaboration is often hindered by faults (e.g., node failures, synchronization errors, etc). Therefore, it is important that the nodes *adapt* to changes in the environment. On the other hand, since the control of the physical phenomenon is time-critical, it is essential that the nodes provide a minimum level of control at all times.

With this motivation, we focus on component adaptation in SANETs (cf. Section V for previous approaches). In component adaptation, the network adapts to the changes in the environment by composing suitable components at run-time. We present an approach for adaptation by systematically using the reset framework [2]–[4]. We apply this framework in Siesta [5] and show how it adapts to node failures by composing a new routing component while providing assurance guarantees.

Organization of the paper. In Section II, we introduce the reset framework. In Section III, we illustrate the framework in the context of vibration control application. Then, in Section IV, we discuss some of the questions raised by this work and in Section V, we discuss the related work. Finally, in Section VI, we make the concluding remarks.

II. DISTRIBUTED RESET FRAMEWORK

In [4], each node is instantiated with the reset framework that contains a *component manager*, a *reset module*, an *adaptation module*, and a *component library* (cf. Figure 1).

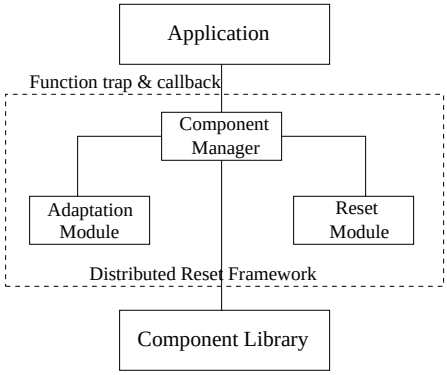

Fig. 1. Framework architecture

Component manager. This module uses *meta-application* and *meta-component* files to define the constraints imposed by the application and the components. The meta-application file details the methods exposed by the component. The meta-component file contains the parameters necessary to compose a new component.

Reset module. Adaptation is performed using two reset waves: (1) *initialization wave* and (2) *replacement wave*. The initialization wave changes the state of each node to *transit* and initializes the new component.

The replacement wave begins with a *transition wave*. Since a component installed at one node may *depend* on components at other nodes, it cannot be replaced independently at each node. Furthermore, the application may be depend on the component. To deal with this problem. each node checks the state of the old component (cf. [4] for more details). Depending on the state, it replaces the old component. Once a node completes replacement and receives *completion wave* from all its children, it propagates the completion wave to its parent.

Adaptation module and component library. The adaptation module has the logic to select the new component. And, the library maintains the components. We refer the reader to [6] for details on these modules.

III. CASE STUDY: VIBRATION CONTROL

We apply the reset framework for routing in Siesta, a simple NEST application simulator [5], for vibration control

experiment. The application consists of 50 nodes aligned along the beam that is subjected to an outside disturbance (cf. Figure 2). Each node has a sensor and an actuator. Sensors measure the point velocity of the beam and the actuators produce point force to counter the vibration. The objective of this application is to control the vibration in response to the disturbance.

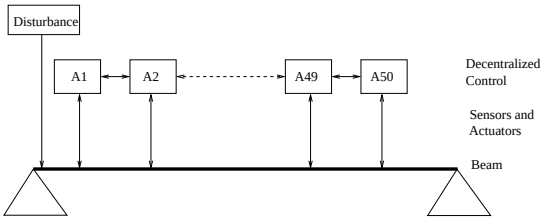

Fig. 2. Simulation setup of beam vibration control experiment

We consider the case where distributed control is used to control the beam vibration in Siesta. In distributed control, each node sends its sensor values to other nodes that are within a specified distance (called the *reach* of the node). Each node then calculates its actuator output based on a weighted average of its own sensor value and the sensor values it received. (The actual algorithm used to determine the actuator output is not relevant for this discussion and hence, is omitted.)

Siesta defines an adjacency relation among the nodes. This relation identifies the nodes that can communicate directly with each other. Communication between non-adjacent nodes is achieved using the static routing information maintained at each node. The default version of the application does not deal with faults, i.e., if a node fails, it affects the distributed control at the nodes that are within the reach of the failed node.

Remark. By node failure, we mean that the router of the corresponding node had failed. We assume that the sensor and the actuator corresponding to a failed node can still participate in the control. In case of failed nodes, the actuator provides only a *local* control (i.e., using only its own sensor value).

A. Dynamic Adaptation of Routing Component in Siesta

As mentioned earlier, in presence of node failures, distributed control of vibration along the beam is affected. To deal with this problem, we use the distributed reset framework to compose a new fault-tolerant routing component with the application at run-time. (Note that the code for new component may reside already in the nodes or disseminated across the network using a code distribution service [7]–[10].) We describe how we specified the new routing component and dynamically replaced it in the application, next.

In Siesta, the *nest.mw.Router.RouteMessage* object creates an action if the router has packets to send. This object uses: *getNextHop*, *setNextHop* and *route* methods. The *route* method uses *getNextHop* to determine the next hop and creates an action that sends the message. The *setNextHop* method is used to setup the routing structure.

In the presence of faults, the framework initiates replacement of the routing component. It uses the meta-application and the meta-component files to verify whether the replacement can proceed.

Meta-application file. The framework uses the meta-application file to determine the methods that need to be trapped. This file also specifies the methods that are exposed by the application to the component. The meta-application file is as follows:

```
1 trapped_class: nest.mw.Router.RouteMessage
2 num_exposed_functions : 1
3 trapped_function_#1: route
4 tf_#1_parameters: Integer, Object
5 tf_#1_returntype: nest.mw.Action
6 app_nexthop_get_function: getNextHop
7 ...
```

Based on this file, the framework traps the method *route*. The application provides *getNextHop* and *setNextHop* methods.

Meta-component file. We developed *AlternateRoute*, a routing component that uses alternate links if the current next hop has failed. It requires that the application provides the *getNextHop* and *setNextHop* methods. The framework instantiates this information in the meta-component file (using the meta-application file). The meta-component file for this component is as follows:

```
1 new_class: nest.ftcomp.AlternateRoute
2 num_new_functions: 1
3 new_#1__function: route
4 new_#1_parameters: Object, Object
5 new_#1_returntype: Object
6 app_nexthop_get_function:
7 ...
```

Dynamic composition. The default routing component in Siesta do not exhibit mutual dependency. Therefore, each node removes its component and adds *AlternateRoute* component independently.[1] The framework traps the calls to the *route* method of *nest.mw.Router.RouteMessage* and invokes the *route* method of *nest.ftcomp.AlternateRoute*.

B. Simulation Results and Analysis

In this section, we study the performance of the vibration control when nodes fail. Specifically, we compare the control provided by the network in response to an exogenous disturbance with and without the distributed reset framework.

In our simulations, the disturbance has a sinusoidal effect on the beam. The sampling rate is 2 KHz and the application supports a *reach* of 5 nodes. Figure 3 shows the output of the simulation (with no failures). The bottom plot shows the sensor readings and the top plot shows the actuation values. (Each bar denotes the sensor/actuator values of a node 1-50, left to right.) We note that actuation values are proportional to sensor readings.

Remark. We note that we focus on the first 15 nodes in our simulations with failures as the vibration is severe in the first half of the beam.

Figure 4 shows the sensor readings and the actuator values after nodes 11 and 12 had failed, with and without adaptation.

[1]If there are dependencies among components added to the application then the components may have to be added in a certain order (cf. [4]). *AlternateRoute* component does not exhibit such dependencies.

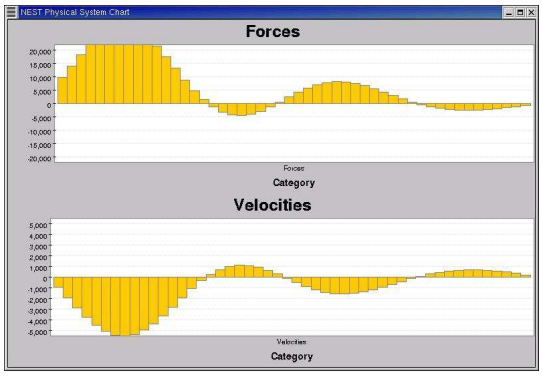

Fig. 3. Beam vibration control output from Siesta

We fail these nodes simultaneously after the network initializes and starts the control application (i.e., after 6000 simulation steps).

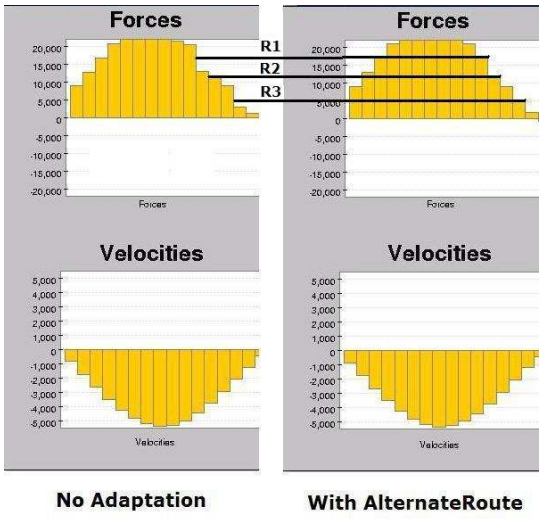

Fig. 4. Beam vibration control output where sensors 12 and 13 have failed

In the absence of adaptation, the actuator values of nodes 8-14 are not proportional to the point velocity readings. In particular, we observe that nodes 8-14 exhibit insufficient control in response to the corresponding point velocities. As mentioned earlier, such anomalies are not desirable in vibration control of fairing structures.

With *AlternateRoute* component, we observe that the control does not exhibit any anomalies. Furthermore, in Figure 4, we identify the difference in control at nodes 10, 11 and 13 in our simulations with and without adaptation (cf. bold lines marked *R1*, *R2*, and *R3* respectively). Specifically, we observe that without adaptation the force output at node 10 (respectively, 13) is around 28% (respectively, 20%) less than the corresponding output with adaptation.

Remark. We performed simulations with failure of more nodes and the results are similar.

Analysis. We now briefly summarize the performance of the adaptation using the distributed reset framework from our simulations.

- *Minimal blocking.* In our simulations, we observed that the replacement of routing component did not block the control application. Specifically, the adaptation was agile and transparent to the user.
- *No localized anomalies.* The simulations with the distributed reset framework exhibited no localized anomalies, i.e., no nodes exhibited insufficient or excess control. The network provided the required level of control at all times. Furthermore, with adaptation, the control improved $20-28\%$ near the reach of the failed nodes compared to the simulation where no adaptation was performed.

IV. Discussion

Now, we discuss the questions raised by this work.

A. Implementation in Resource Constrained Sensors

Since the adaptation module determines the suitable components, we can have this module only at the base station. And, the developer can maintain the library in the secondary storage or use a code distribution service (e.g., [7]–[9]) to disseminate the new component. The reset module implements a simple diffusing computation to ensure correct adaptation. The component manager checks whether the adaptation can proceed correctly, which can be also be done offline.

B. Adaptation in Large Scale Sensor Networks

First, we note that large scale networks are usually partitioned into multiple sections; each section has at least one node that is responsible for data collection and communication with the outside world. The reset operation can be performed in each section independently and simultaneously. Furthermore, the adaptation does not block the entire network. The new component can start executing even before the old component is removed. Since the framework ensures synchronization, simultaneous execution of the old and the new components do not result in invalid computations.

C. Scope of the Framework in Sensor Networks

As mentioned in Section IV-A, we note that the framework shown in Figure 1 can be instantiated in resource constrained sensors. Furthermore, in [4], [11], authors have shown that the framework ensures the following: (i) *atomicity*, i.e., all the nodes in a distributed system are replaced with a new component or none are replaced, (2) *minimal blocking*, i.e., the blocking introduced at the application during adaptation is minimal, and (3) *synchronization*, i.e., a node that has instantiated the new component does not interact with the node that has not yet added the corresponding component. Thus, the framework ensures *correct* adaptation, which has been formally specified and verified in [11].

The case study considered in this paper is relevant in the sense that it clearly shows the advantages of using the framework in sensor networks. Specifically, it shows the following: (1) the application is not blocked during adaptation, and (2) synchronization is achieved quickly. We note that even if the application had to be blocked in order to ensure correct

adaptation, the framework does not require the entire network to stop. The results presented in Section III-B reinforces these claims. Based on the results presented in this paper and mechanism to verify the correctness of adaptation (cf. [11]), we expect that the framework has the potential to be applicable in sensor networks.

V. Related Work

Work on adaptation in SANETs have focused on three major areas: parameter adaptation, code distribution, and support for adaptive software.

A. Parameter Adaptation

In [12]–[14], data collection protocols are proposed that adapt to the amount of energy left at the neighbors. However, in parameter adaptation, it is not possible to use components or algorithms that were left unimplemented at the time of the original design of the application. By contrast, we concentrate on dynamically replacing a component in the application.

B. Code Distribution

In [7]–[9], network reprogramming protocols are proposed to reprogram the network with a new application. However, since reprogramming is a time-consuming and a communication-intensive operation, these approaches do not meet the requirements of time-critical applications.

In [10], a role based code distribution is proposed for sensor networks that uses cross-layer information about the role assigned to a node in deciding whether a node needs a code update or not. Also, the paper proposes a *data management framework* that determines the best components and, thus, adapts to changing parameters and requirements. However, in [10], synchronization across components installed at different nodes and the dependency relation are not handled.

C. Adaptive Software in Sensor Networks

In [15], [16], approaches for supporting automated software adaptation in sensor networks are proposed. In [15], the authors propose an XML based approach to exchange and manipulate descriptions of components. However, they do not provide assurance guarantees about adaptation. In [16], unlike the reset framework, the authors focus on providing an infrastructure to exchange software extensions.

VI. Conclusion

In this paper, we presented an approach to implement distributed reset based dynamic adaptation in SANETs. We studied the performance of such adaptation in beam vibration control experiment in the Siesta platform. We showed that the framework allows the application to dynamically replace a routing component. The simulation results showed that the adaptation is minimally blocking. This result is desirable in SANETs as the application is time-critical. Also, we showed that with the reset framework improved the control by around $20-28\%$ in the neighborhood of the failed sensors (compared to the case where no adaptation was performed).

We expect that the reset framework facilitates the verification of adaptation and ensures correctness. Towards this end, in [11], the authors propose an approach to verify adaptation (achieved using the reset framework). Thus, we can leverage on existing techniques to verify the correctness of adaptation in SANETs. Furthermore, in [17], the authors propose an approach for building a family of components that provide similar functionalities. We expect that such an approach would be applicable in designing components for SANETs.

Acknowledgments. This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF EIA-0130724, and Michigan State University.

References

[1] "Challenge problem description for networked embedded software technology (NEST)," April 2002, The Boeing Company, P. O. Box 516, St. Louis MO 63166.

[2] A. Arora and M. G. Gouda, "Distributed reset," *IEEE Transactions on Computers*, vol. 43, no. 9, pp. 1026–1038, 1994.

[3] S. S. Kulkarni and A. Arora, "Multitolerance in distributed reset," *Chicago Journal of Theoretical Computer Science*, 1998.

[4] S. S. Kulkarni, K. N. Biyani, and M. Arumugam, "Composing distributed fault-tolerance components," *Workshop on Principles of Dependable Systems*, June 2003.

[5] A. Ledeczi, M. Maroti, and I. Bartok, *SIESTA - Simple NEST Application Simulator Siesta v1.0*, Institute for Software Integrated Systems (ISIS), Vanderbilt University, October 2001.

[6] K. N. Biyani, "Dynamic composition of distributed components," Master's thesis, Computer Science and Engineering, Michigan State University, December 2003, available at: http://www.cse.msu.edu/~biyanika/thesis.

[7] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," *In Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2004.

[8] S. S. Kulkarni and L. Wang, "MNP: Multihop network reprogramming service for sensor networks," *International Conference on Distributed Computing Systems*, 2005.

[9] S. S. Kulkarni and M. Arumugam, "Infuse: A TDMA based data dissemination protocol for sensor networks," *International Journal of Distributed Sensor Networks*, March 2006.

[10] P. Marrón, D. Minder, A. Lachermann, and K. Rothermel, "TinyCubus: An adaptive cross-layer framework for sensor networks," *Information Technology*, 2005.

[11] S. S. Kulkarni and K. Biyani, "Correctness of component-based adaptation," *Component-based Software Engineering (CBSE)*, May 2004.

[12] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," *In Proceedings of the International Conference on Mobile Computing and Networks (MobiCom)*, August 1999.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A scalable and robust communication paradigm for sensor networks," *In Proceedings of the International Conference on Mobile Computing and Networks*, pp. 57–67, 2000.

[14] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian, "Adaptive target tracking in sensor networks," *In Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS)*, January 2004.

[15] H.-C. Kim, H.-J. Choi, and I.-Y. Ko, "An architectural model to support adaptive software systems for sensor networks," *In Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 670–677, 2004.

[16] G. Alonso, "PROSE: Automated software adaptation for pervasive computing," *European Research Consortium for Information and Mathematics*, no. 58, July 2004.

[17] K. Biyani and S. S. Kulkarni, "Building component families to support adaptation," *In Proceedings of the International Workshop on Design and Evolution of Autonomic Application Software (DEAS)*, May 2005.