An Experimental Facility for Cross-layer Adaptation of Service Oriented Distributed Systems

Shanshan Jiang, Svein Hallsteinsen, Arne Lie
SINTEF ICT, Postboks 4760 Sluppen, 7465 Trondheim, Norway
{Shanshan.Jiang, Svein.Hallsteinsen, Arne.Lie}@sintef.no

Abstract

Ubiquitous and IoT (Internet of Things) systems consist of many parts, are highly distributed and need to be adaptive in a highly dynamic environment. The exploitation of adaptation possibilities at different layers needs to be coordinated to get an optimal result. However, it is difficult to test and evaluate such distributed systems with regard to their adaptive behaviour. This paper presents the design and implementation of a hybrid simulation based experimental facility for cross-layer adaptation of such adaptive systems. It is based on adaptation logic that builds runtime adaptation models based on information from the application, communication and hardware layers and uses the model for coordinated adaptation of these layers. The simulation facility has been implemented based on the MUSIC adaptation framework. Our work has been inspired by use cases from the subsea sensor networks and ambient assisted living domains, and the simulation facility is being applied to study the benefits of cross-layer adaptation in these domains. As preliminary validation of the proposed approach we discuss initial experience from the subsea sensor network use case. However, we believe that such simulation facility is generally applicable for application domains exhibiting highly distributed systems in heterogeneous and dynamically varying environments.

1. Introduction

The advances of sensor technologies and the prevalence of ubiquitous computing have brought about numerous innovative applications. In particular, they contribute to the vision of the IoT (Internet of Things) that any object from both the real world and the virtual world is connected anytime, at anyplace and interacts with anything. Such ubiquitous/IoT systems are complex, i.e., they have many parts, are highly distributed, and execute in a dynamically varying computing and communication environment. Adaptation capabilities are needed for such systems to adapt to changing environment conditions and changing application requirements, such as dynamic discovery of available resources and services, and dynamic reconfigurations and bindings resolved at runtime. Various adaptation capabilities and mechanisms have been proposed at different layers such as application, service interfaces, network and hardware layers [1,2,3], but they mainly work in isolation. In recent years, the importance of *cross-layer adaptation* enabling coordinated adaptation across layers has been recognized and received more and more attention in order to maximize the benefits from adaptations at different layers [4,5,6,7].

We are interested in studying how adaptation middleware can contribute to the development, deployment and evolution of such systems. However, setting up a test system of realistic size and making the environment behave in ways that allow systematic testing is difficult. In some contexts testing in a simulated environment is the only option. For instance, it is practically infeasible to deploy sensors running adaptive middleware to a real underwater sensor network due to both technological and economic constraints. Furthermore, our previous experience from working with adaptive systems has shown that there are strong needs for tools that can demonstrate or visualize the adaptive behaviour of systems under various contexts and "controlled" environment during development and testing.

This paper was presented at the NIK 2011 conference. For more information see http://www.nik.no/

Hybrid simulation, i.e., a simulation facility that can run a mixture of real application software components and simulated components for applications as well as communication and physical environment, appears as a good approach. Our work on a hybrid simulation facility is an attempt to address these two tightly coupled concerns: to support cross-layer adaptation and be able to test and evaluate the adaptation behaviour of highly distributed service oriented adaptive systems. In this paper we present the design and implementation of such hybrid simulation facility. The work is based on the MUSIC [8] adaptation framework. The idea is to use an environment simulator to simulate the physical environment and communication between nodes. The simulator receives events that simulate the dynamics of the target environment as well as messages generated by other nodes in the system and feeds them to the MUSIC middleware, which then performs adaptation and reconfiguration accordingly. This hybrid approach combines simulation and real execution, where real MUSIC middleware instances (adaptation logic) are running and adapting real applications on top of a simulated network and hardware layer.

The simulation facility enables cross-layer adaptation by allowing the adaptation logic to control adaptations at the application, network and hardware layers. Our approach is based on a centralized Adaptation Manager, which makes adaptation planning based on information obtained in different layers and controls the adaptations at these layers.

MUSIC focused primarily on the adaptation of applications. Other layers were considered more like context, and used ad-hoc facility to control them. This paper provides a complete and unified model for cross-layer adaptation and a simulation facility for testing this approach. The main *contributions* of this paper are therefore the generalisation of the capabilities of the MUSIC framework to support cross-layer adaptation and a hybrid simulation facility suited to studying the benefits of cross-layer adaptation, and to testing and tuning the adaptation logic. Although the simulation facility is implemented based on the MUSIC middleware, the cross-layer adaptation model is sufficiently general to serve as a basis for standardising layer interfaces supporting cross-layer adaptation.

The rest of the paper is organized as follows: Section 2 presents some related work about simulation and cross-layer adaptation. Assumptions and requirements for the simulation facility are given in Section 3. Since the motivation is to test and tune adaptation logic and study its effect, an adaptation framework needs to be explained. Section 4 serves this purpose by briefly describing the main concepts in the underlying adaptation framework. The model for cross-layer adaptation is described in Section 5. Section 6 presents the simulation architecture and how the simulation facility works. Section 7 reports our initial experience with the subsea use case and gives some discussion. Finally, Section 8 summarises the conclusions and outlines future work.

2. Related Work

Following the methodologies of design science, systems designed need to be validated and tested. Software simulation has been widely used in this regard. A simulator can be either completely tailor programmed, or one can use some kind of simulator or programming framework. Examples on the latter are e.g. Matlab[1], which is most commonly used for simulating continuous time processes, such as the physical layer of communication systems. This can include signal modulation and demodulation, forward error correction, adding channel impairments such as attenuation, distortion and noise,

[1] http://www.mathworks.com/products/matlab/

and signal detection, equalization and synchronization. Such computations are however CPU demanding. Higher layers such as MAC (medium access control) and network layer (routing and link connections for packet switched networks) are therefore usually simulated with *discrete event-based* techniques. In such simulators, only the time instances where packets are created, sent, and received, are counted for. Physical impairments for wireless communication are in the latter cases simplified to e.g. a static signal strength or signal-to-noise value, giving e.g. a specific link bit rate, energy per bit needed to transmit, and the link propagation delay. Examples of such simulator frameworks are ns-2[2], ns-2-miracle[3], WOSS[4], ns-3[5], OMNeT++[6], and SimPy[7]. In these systems, a rich library of generic purpose modules are included, ready for use, such as the TCP/IP protocol, routing protocols, WLAN MAC protocol, and signal attenuation characteristics. FTP downloads and constant bit rate sources are typically present in a small set of applications to generate data payload. The user is left with the job of defining a static or dynamic network topology, and can easily scale the number of communication nodes and the traffic load itself. The user should run each simulation topology a certain number with random seeds giving stochastic fluctuations where appropriate, so as to obtain qualified results with averages and confidence intervals. Our subsea case study uses WOSS to obtain simulated network conditions.

However, existing simulation or simulators focus mainly on MAC, network layer, and transport layer, and very seldom supports an application layer with *adequate behaviour*. Special attention must be given when application adaptation is in use, because of the online interaction between network state and application state [9]. Our work intends to provide a solution to this challenge.

Various approaches for cross-layer adaptation integrating different layers have been proposed. Gjørven et al. propose a framework for cross-layer adaptation based on QuA middleware by integrating interface and application layer mechanisms [4]. The work of Popescu et al. builds cross-layer adaptation on the taxonomy-based event-driven discovery and selection of adaptation templates, which define the behaviour of adaptation logic as BPEL processes [5]. Schmieders et al. propose an approach that instruments cross-layer adaptation to avoid SLA violations. It exploits multiple adaptation mechanisms available on all SBA layers and chooses the right mechanism based on an adaptation strategy [6]. Yuan et al. present their design and evaluation of a cross-layer adaptation framework for mobile multimedia systems called GRACE-1 [7]. Their framework supports coordinated adaptation in the hardware, OS and application layers with mechanisms for global and local adaptation. Vidackovic et al. propose a generic, cross-layer monitoring and adaptation framework based on a top-down approach, focusing on the business model and its interactions with regard to the lower layers [10].

In contrast, our approach models the variation points (i.e. things that can change/control) at different layers and builds runtime adaptation model based on them for adaptation planning to achieve coordinated adaptation across layers.

---

[2] http://isi.edu/nsnam/ns/
[3] http://www.dei.unipd.it/wdyn/?IDsezione=3966
[4] http://telecom.dei.unipd.it/ns/woss/doxygen/
[5] http://www.nsnam.org/
[6] http://www.omnetpp.org/
[7] http://simpy.sourceforge.net/

3. Assumptions and Requirements for the Simulation Facility

To derive the assumptions and requirements for the simulation facility we have investigated two use cases, one from the subsea environmental monitoring domain and one from the AAL (Ambient Assisted Living) domain as examples of typical ubiquitous computing and IoT systems. The subsea environmental monitoring use case is used in this paper to illustrate the ideas.

We consider that the target environment is characterised by *large numbers of heterogeneous sensors, heterogeneous communications infrastructure as well as heterogeneous service publication, discovery and binding technologies*. Sensor networks are typically constrained by bandwidth, processing power, memory and power consumption. For example, underwater sensor networks pose big challenges for the communication infrastructure. Such networks need to rely on acoustic communication since radio signals propagate very poorly in sea water. In addition, radio communication is used from surface to land, either directly or via satellite, depending on the distance. Variations in weather, sea temperatures and salinity strongly influence the communication conditions under water and the use of mobile units (ships on the surface, autonomous underwater vehicles - AUVs - and gliders under water) adds to the variation. Such environment is characterized by the low capacity and long delays of the acoustic channel, the potentially huge number of sensors, and the energy constraints of battery powered underwater units. Furthermore such systems typically involve a wide spectrum of devices, from tiny sensors through buoys and AUVs to nodes installed on ships, and thus heterogeneity both of devices and communication links is an issue. One special design choice for such resource constrained environment is to support UDP. Such environment usually prefers UDP over TCP, since UDP has less overhead and is ideally suited to short transactions.

From this we derive several requirements for the simulation facility. Firstly, the simulation facility needs to support context aware and adaptive applications based on SOA (Service Oriented Architecture) architectural style running on resource constrained environment. Such applications consist of many parts collaborating by providing services to and using services from each other and application configuration forms dynamically and adapts to changes in the environment. Examples of changes include 1) nodes and services appears and disappears; 2) resources become exhausted (e.g. batteries running flat), 3) communication links appear and disappear and vary in capacity and other characteristics.

Secondly, it needs to be based on a hybrid simulation approach. There will be a mixture of real software application components and simulated components. Based on the separation of application and adaptation logic, it will use the skeleton of application components, e.g., based on the MUSIC approach, so that we can test the adaptation behaviour at an early stage independent of business logic.

Thirdly, the simulation facility should be general, not only suitable for subsea and AAL systems, but more generally for M2M/IoT type of systems, where alternative networks (e.g. WLAN, Bluetooth and UMTS/GPRS) and fluctuations in availability and capacities are common and adaptation to them is required. This also means that the simulation facility should work with heterogeneous discovery and binding technologies.

Finally and very importantly, the facility should support cross-layer adaptation. This means that the adaptation framework should integrate adaptation capabilities in different layers to enable coordinated adaptation across layers. For instance, alternative links between a pair of nodes may exist in cases like: 1) multi-hop vs. single-hop route in wireless radio sensor networks or underwater acoustic networks (communication using multiple short hops may be energy and bandwidth beneficial compared to a single long

hop); 2) in radio communication, there can be multiple frequency bands and standards available, such as WLAN, Bluetooth and UMTS/GPRS; and 3) in the extreme end, in the cognitive radio network vision, the terminals have powerful front-ends and digital intelligence to dynamically sense and adapt and select vacant frequency spectrum at a fine temporal granularity [11]. It would be desirable that adaptation at the communication layer based on the availability and properties of such alternative links can be coordinated with the adaptation at the application layer.

4. Adaptation Framework

This work builds on the MUSIC adaptation framework. The MUSIC framework is based on an externalized approach to self-adaptation where the adaptation logic is delegated to generic *middleware* working on the basis of *models* of the software and its context represented at runtime [8,12].

Systems are conceived as collections of *applications* collaborating by providing *services* to and using services from each other, and executing on a network of computer nodes connected by communication links. Each node runs an instance of the adaptation middleware which is responsible for adapting the applications hosted there. A *variability model* associated with the application, specifies its adaptation capabilities in terms of *variation points,* and how the alternative bindings of the variation points affects its *properties*. Supported adaptation mechanisms include component replacement, service binding and varying the quality level of provided services. Variants are characterized by properties which vary between their variants. Such variants can be components or services provided by external service providers. These *properties* express functional and/or extra-functional (i.e. QoS) properties of the component or provided service and are modelled by *property predictor* functions. The properties and the resource needs of an application variant are computed by predictor functions based on the properties and resource needs of the included component and service variants. Such property annotation of the architecture model has to be provided by the developer.

The variability model defines a *utility function*, which expresses how well suited a given configuration is in a given situation based on the predicted values of the varying properties, the properties of current context and resources, and service level agreements with consumers of provided services.

The adaptation middleware monitors relevant context and resources, components installed on the node it runs on and services available in the environment. When significant changes occur adaptation planning is triggered. The middleware evaluates the utility of all configurations satisfying the resource constraints and selects the one with the highest utility. If this is different from the currently running configuration it reconfigures the application to the selected configuration. One instance of the middleware controls the adaptation of the set of application running on the node it runs on and coordinates their adaptation so as to maximise the overall utility. Coordination of the adaptation on collaborating nodes is achieved through the dynamic discovery and binding of services and settling of service level agreements between applications on different nodes.

Variability models are represented at runtime as *plans.* A plan contains a recipe for instantiating a component realization and a function for predicting its QoS properties in the form of property predictors. The dependency on an external service is represented as a *service plan*, with an associated set of *service plan variants* representing the available service providers.

The MUSIC middleware is based on a pluggable architecture and implemented on top of OSGi [8]. Figure 1 gives an overview of the middleware architecture. Plans and

plan variants are stored in the *Plan Repository* in the *Kernel*. The *Context & Adaptation Middleware* handles the adaptation planning process, which is triggered by context changes detected by *Context Sensors*. The *Adaptation Manager* builds valid application configurations by solving their dependencies, ranks them by evaluating their utility based on the computation of the predicted properties, and selects the configuration that provides best utility. The *Configurator* handles the reconfiguration process using the configuration and plans selected by the Adaptation Manager. The *Communication* provides basic support for SOA in distributed environment. The *Service Discovery* publishes and discovers services using different discovery protocols. Whenever a service is discovered, a corresponding service plan variant is created in the plan repository. The *Service Binding* is responsible for the binding and unbinding of services. At the service provider side, it exports services (i.e. enable them to accept service requests), and at the service consumer side, it provides bindings (i.e. remote access) to the discovered remote services. The *Negotiation Framework* is an optional component responsible for service level negotiation and violation handling. It interacts with the Context & Adaptation Middleware and the Communication to realize the adaptation process integrated with service level agreement mechanism [13].

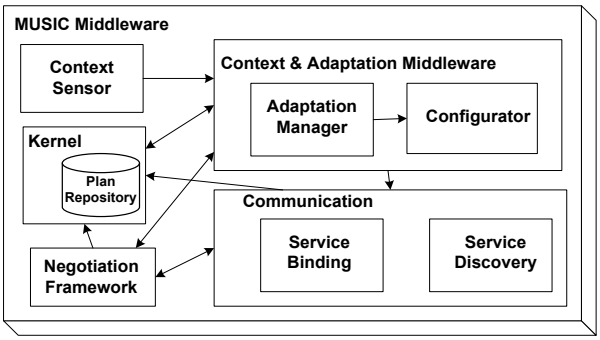

Figure 1 Overview of MUSIC middleware platform

5. A Model for Cross-layer Adaptation

The MUSIC adaptation framework was primarily built to do adaptation at the application layer although information about the communication and hardware layers are also taken into account as context. In order to realize cross-layer adaptation, adaptation capabilities of the communications and hardware layers also have to be exploited and adaptation at all three layers must be coordinated in a way that optimises the overall working of the systems. To achieve this we propose an extension of the MUSIC framework enabling the Adaptation Manager to do such coordinated adaptation planning and reconfiguration across all three layers.

5.1 The Unified Model

Our approach is based on a unified model for adaptation (Figure 2). The main idea is that the communication and hardware layers also expose their adaptation capabilities in the form of a variability model, defining variation points with sets of variants.

- At the *hardware* layer, variation points can be configuration parameters for a hardware component, such as the clock speed of CPU, battery power on/off, memory bank turning on/off. Context information includes resource usages, e.g., battery level, CPU and memory usage.

- At the *communication* layer, there can be alternative links as illustrated in Section 3 with varying link properties. Examples of *link properties* are delay, bit rates, transmission power, energy consumption and forwarding cost.
- At the *application* layer, examples of variation points are different component realizations for a component type and service offerings from different providers with varying quality, which may dynamically appear, change and disappear (cf. Section 4).
- In addition, *user context* (e.g., home, office or driving) may influence adaptation at all layers and shall be considered in the adaptation planning.

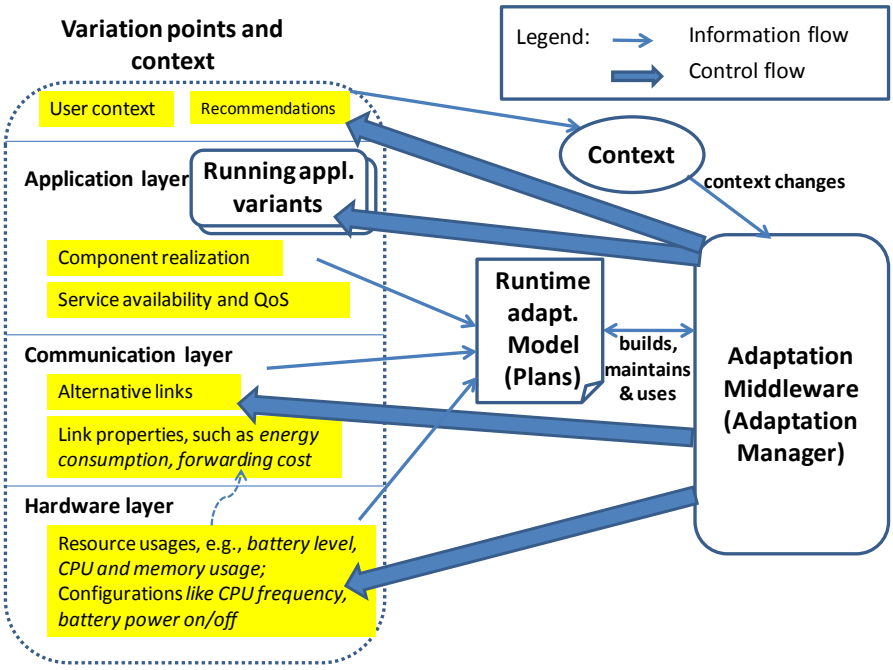

Figure 2 Cross-layer adaptation framework

The Adaptation Manager uses information about the variation points to build a *runtime adaptation model* (represented as *plans*), which describes what can be changed and controlled at runtime. The model will be dynamically updated when the variants change. The adaptation is triggered by context changes. The Adaptation Manager controls the adaptation at the application layer by selecting and reconfiguring to the best application variants (i.e., binding to the selected component and service instances). By selecting the specific communication link that can contribute to the best utility, e.g., with lowest forwarding cost based on the battery levels and energy consumptions, the Adaptation Manager also controls the adaptation at the communication layer. The Adaptation Manager can also select configuration parameters of a hardware resource, and ask the hardware to turn on/off memory bank or power on/off battery, thus controls the adaptation at the hardware layer as well. Adaptation middleware may also provide *recommendations* to user, e.g., to close an application due to battery running low.

5.2 Incorporating Link Properties in Cross-layer Adaptation

Link properties are typically provided by network sensors in real deployments or by network simulators in simulations. In particular, we define a special link property called *forwarding cost* to represent the resource usage of a link outside the source and target nodes. For example, we may simply represent the forwarding cost of a multi-hop link as the average value of the battery levels for all the relay nodes. This means that resource

usages of other nodes can be passed over to higher layers as a special link property via the communication layer. Figure 3 illustrates this idea with an example of multi-hop routes as alternative links from the subsea sensor network use case. In the figure, there are two alternative links between **A** and **B**. The link properties considered including transmission power (*trPwr*), which in both cases are the power used by node **A** to send messages; forwarding cost (*fwdCost*), which is zero for direct link and a function of the average battery level (*pwrLvl*) of intermediate nodes for multi-hop link.

In order to include communication layer in cross-layer adaptation the link properties should be reflected in the representation of the discovered services, which in MUSIC terms, is called service plan variants. We have modified/extended the *Service Discovery* and *Service Binding* components of the MUSIC middleware to include the adaptation of communication layer in the coordinated adaptation. The approach is to consider link properties as special properties for discovered services and annotate the service plan variants for discovered services with link properties of available links. Figure 3 shows two different service plan variants (**S**, $\mathbf{l_1}$), (**S**, $\mathbf{l_2}$) created at node **B** with properties of available links ($\mathbf{l_1}$ and $\mathbf{l_2}$) when node **B** discovers the service **S** provided by node **A**. These service plan variants (therefore the link properties) are then used by the Adaptation Manager in its adaptation planning (cf. Section 4). When new configuration is selected after adaptation the service plan variants selected will decide which link to use for communication. In this way the Adaptation Manager also controls the adaptation of the communication layer.

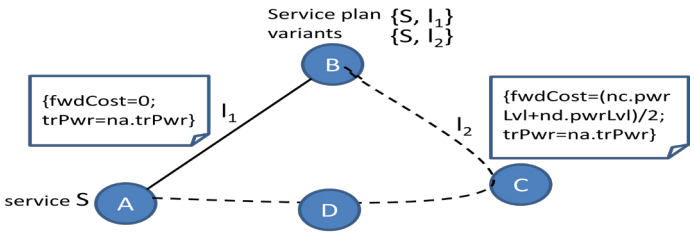

Figure 3 Nodes, services and alternative links annotated with link properties

6. Design and Implementation of the Simulation Facility

The simulation facility is based on a hybrid simulation model with a discrete event-driven environment simulator to simulate events from any layer in the physical environment. The environment simulator gets event-based input and notifies the MUSIC middleware running on corresponding node(s) about the changes. The MUSIC middleware then performs adaptation and reconfiguration accordingly.

Due to the hybrid nature, the simulator has two types of inputs and they have different impact on the *energy consumption* which is an important property in the evaluation of system performance from the adaptation aspect. On the one hand, the simulator may receive *simulation events*, e.g., events from communication layer generated using other network simulators, which simulate target network conditions (such as the network topology, the characteristics of the communication links and their changes). These simulation events are not real messages sent over the actual networks and thus have no direct impact on the energy consumption for the receiving nodes. On the other hand, the simulator may receive *messages* sent by other nodes in the system. These are the *real* messages needed for running the adaptive systems, thus need to consume energy when sending and receiving the messages.

The simulation architecture is depicted in Figure 4. The ProSUS environment simulator (named after the internal project ProSUS) reads the simulation events and simulates the communication between several MUSIC nodes. Each node is represented

as an OSGi/JVM (Java Virtual Machine) instance with the appropriate middleware and application bundles installed. Every communication is via the ProSUS environment simulator.

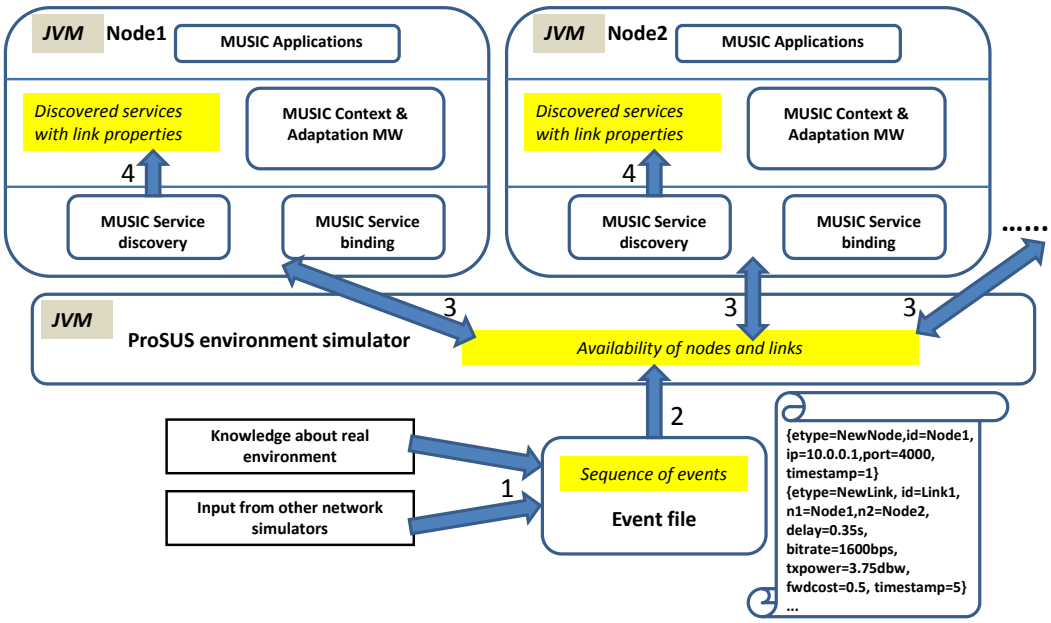

Figure 4 Simulation architecture

Some more details about the working of the environment simulator and its interaction with the middleware are explained in the following (the numbers in parenthesis refer to corresponding numbers in Figure 4):

- An event file is generated containing sequence of simulation events based on knowledge about the real running environment (1). In general, any event from each layer can be included. For the hardware layer, events can be failure of hardware (e.g., memory bank), or battery level depleted. For the communication layer, events can be new node, dead node, new link, broken link and change of link properties. Events can also be contexts set by user, e.g., user can switch the context from "driving" to "in-a-meeting" during simulation. Our current implementation covers only events from the communication layer, which are generated based on knowledge about real communication layers and input from other network simulators.
- The ProSUS simulator has an *event handler* which reads the events from the simulation event file in the temporal order given by the event timestamps, and maintains a table of nodes and links in the network and their properties (2).
- The changes of nodes and links are sent to the MUSIC middleware by the simulator and trigger the middleware for service discovery (3).
- The Service discovery and binding components discover the services over the available links (3) and create service plan variants for each service over an available link (4). The link information is considered as an explicit property of a service (cf. Section 5.2). These service plan variants are used by the MUSIC middleware for adaptation and reconfiguration.
- The MUSIC middleware uses the ProSUS environment simulator for sending and receiving messages required for service discovery and communication between nodes (i.e. service binding). In other words, the ProSUS environment simulator also transmits MUSIC messages via a middleware selected link according to the link properties stored in the table. For example, it can delay the message transmission

according to the link delay property or drop packets according to the packet loss property.

7. Experience and Discussion

7.1 Initial Experience with the Use of the Simulation Facility

As the first application of the simulation facility we have developed a scenario for the subsea environmental monitoring use case [14]. In this scenario we identified several sources of variation: 1) *communication capacities and energy consumption*: Acoustic communication in sea water implies a highly limited transmission capacity, long delays and a high ratio of energy per bit. These parameters vary over time and with the depth and are influenced by environmental phenomena that typically have a cycle of *one year*; 2) *owner needs*: Under normal condition, the requirement to sampling frequency and freshness of data may be relaxed, while in the presence of accidents causing oil spill or leakage of radioactive material, much more frequent and recent observations are needed; 3) *mobile nodes*: The use of mobile nodes such as ships, AUVs, and gliders to relay the measurements to the onshore installation, causes dynamic variation in the topology of the network. Battery depletion and other node failures have the same effect.

The purpose of this exercise is to study the benefits of cross-layer self-adaptation in this kind of systems, in particular the effect on resilience and energy efficiency. The simulation facility is the natural choice for testing and evaluation of such application as it is practically infeasible to test it in real deployment.

We have used WOSS[8] simulator to obtain link properties. Since phenomena that influence underwater acoustic communication typically have a cycle of one year, we run WOSS simulations for different seasons and topologies and get information about link properties, such as node distance, transmission power, path loss, bit rate and relay nodes for multi-hop links. Properties like *forwarding cost* and *link propagation delay* have to be computed separately based on the above information. Then we manually generate the *event file* with a sequence of events derived from the WOSS simulation output. A typical event is the change of link properties due to seasonal variations derived by comparing the output from different seasons. The timestamp of each event and information about node and link properties are also collected. Figure 4 shows an excerpt of such an event file.

We need to simulate events and data at other layers as well. We need to simulate input to the application such as measurements made by the sensors. We also need to simulate context events such as changed preferences, e.g., oil spill requiring more frequent measurements over a period, and hardware layer events like battery level depleted and memory bank failure. To study the energy effect we need to simulate the control of physical layer, e.g., turning on/off a memory bank or power on/off battery.

We are in the process of implementing the example system based on selected scenarios and plan to execute them in the simulation facility. The simulation of the communication layer has been implemented while other layers have been designed. Even in this early stage, the simulation facility has demonstrated its flexibility by allowing us to simulate the long-term (i.e. yearly) variation in a manageable environment.

[8] WOSS is an extension to ns-2-miracle, aided towards underwater sensor network simulation, where actually a very sophisticated physical layer model is interfaced (the ray tracing model Bellhop), which is run at simulation start-up to provide close to real-life signal-to-noise ratios of all possible network links.

7.2 Discussion

In Section 3 we stated a number of requirements for the simulation facility. Regarding the first requirement, the simulation facility is based on MUSIC, which supports context aware and adaptive applications and is already SOA-based and targets resource constrained environments. Our implementation of communication based on UDP reflects this constraint as well. Regarding the second requirement, the simulation facility allows for applications to run based on various simulated events from the environment as well as to simulate controls at different layers. Even some application components or input can be simulated to allow for early testing. As for the third requirement, we have implemented service bindings based on UDP and RESTful style [15], which are suitable for working with emerging standards and protocols targeted for resource constrained environment, such as CoAP[9]. Finally, we have proposed a unified model for coordinated adaptation of application, communication and hardware layers. We have implemented the core of the simulation facility as described in the paper with support for events at communication layer[10]. It can be easily extended to support events from other layers.

During the design of the simulation facility we faced several design choices. Firstly, there are alternative approaches to cross-layer adaptation. This paper adopts a *centralized* approach, where an Adaptation Manager controls the adaptation of application, communication and hardware layers based on information obtained from different layers. With this approach it is easy to ensure the coordination across layers. The drawback is it may suffer from scalability problems, caused by the combination of variation points from several layers. Alternatively, a *distributed* approach could be used where application, communication and hardware layers are controlled by mechanisms at different layers; these adaptation mechanisms are then governed by high level policies. This approach scales better, but the challenges are the specification of proper high level policies and strategies to ensure the coordinated adaptation, and a solution for conflict resolution.

Secondly, there are also alternative approaches to modelling link properties at the adaptation layer. We have implemented a simple approach by adding link quality information directly to service plan variants which are used for adaptation planning. Alternatively, we might have extended the MUSIC model to represent link properties explicitly in the model, by modelling connections as variation points. This is a more general approach, but it was discarded because it requires substantial changes to the MUSIC tools and middleware.

The use of the simulator requires the construction of an event file with a sequence of events which is representative for the target execution environment of system being studied. This can be obtained in different ways. We have reported how we generate event files manually in the subsea use case. In other cases, such as AAL, we may collect link properties using other network simulators and generate event sequences (semi-) automatically.

8. Conclusions and Future Work

We have presented the design and implementation of an experimental facility based on hybrid simulation for adaptive systems. We also report initial experience from the subsea sensor network use case as preliminary validation of the proposed approach. By

[9] CoAP: IETF draft. http://tools.ietf.org/id/draft-shelby-core-coap-00.txt

[10] The source code for the simulation facility is available at: https://svn.berlios.de/svnroot/repos/ist-music/music-middleware/branches/NetworkSimulator.

separation of application and adaptation logic, the simulation facility allows for testing the adaptation behaviour at an early stage independent of business logic. Using simulation facility allows for instantiating many nodes as needed in the real systems without needing access to the physical equipment and end users in real life setting, giving good potential for scalable testing.

This paper presents a unified model for cross-layer adaptation. The current implementation of the ProSUS environment simulator covers only events from the communication layer so far. We plan to extend it to simulate the effects of configuring the CPU and memory of the nodes as well as to allow users to set contexts for simulation. We are also in the process of implementing applications for use cases from AAL domain and subsea environmental monitoring and test them on the simulation facility. One of the goals is to investigate whether self-adaptive capabilities of sensor systems can save energy and extend sensor lifetime. Another direction is to run large scale simulation on a cloud server, e.g., one such facility as provided by Telenor ASA.

Acknowledgement

This work is partly supported by SINTEF and Telenor through their strategic research cooperation agreement FI-M2M project, as well as MODERATES and ProSUS projects.

References

1. Batista T. V., Joolia A., and Coulson G. *Managing Dynamic Reconfiguration in Component-Based Systems*. In EWSA 2005, LNCS 3527, Springer, 2005, 1-17.
2. Flinn J., de Lara E., Satyanarayanan M, Wallach D., and Zwaenepoel W. *Reducing the energy usage of office applications*. In Proc. Of Middleware 2001, Heidelberg, Germany, Nov. 2001.
3. Corner M., Noble B., and Wasserman K. *Fugue: time scales of adaptation in mobile video*. In Proc. of SPIE Multimedia Computing and Networking Conference, San Jose, CA, Jan. 2001.
4. Gjørven E., Rouvoy R., & Eliassen F. *Cross-layer Self-adaptation of Service-oriented Architectures*. In Proc. of MW4SOC, 37-42, 2008.
5. Popescu R., Staikopoulos A., Liu P., Brogi A., & Clarke S. *Taxonomy-driven adaptation of multi-layer applications using templates*. In Proc. of SASO, 213-222, 2010.
6. Schmieders E., Micsik A., Oriol M., Mahbuk K., & Kazhamiakin R. *Combining SLA Prediction and Cross Layer Adaptation for Preventing SLA Violations*. 2nd Workshop on Software Services: Cloud Computing and Applications based on Software Services, 2011.
7. Yuan W., Nahrstedt K., Adve S. V., Jones D. L., & Kravets R. H. *Design and Evaluation of a Cross-layer Adaptation Framework for Mobile Multimedia Systems*. MMCN, 2003.
8. Rouvoy R., et al. *MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments*. In: Cheng, B.H.C., et al. (eds.) Self-Adaptive Systems, LNCS 5525, Springer, 2009; 164-182.
9. Lie A. and Klaue J. *Evalvid-RA: Trace Driven Simulation of Rate Adaptive MPEG-4 VBR Video*. ACM/Springer Multimedia Systems Journal, Nov. 2007.
10. Vidackovic K., Weiner N., Kett H., and Renner T. *Towards business-oriented monitoring and adaptation of distributed service-based applications from a process owner's viewpoint*. In ICSOC/ServiceWave workshop, 385-394, 2009.
11. J. Mitola III J., G. Q. Maguire Jr. *Cognitive radio: making software radios more personal*. IEEE Journal on Personal Communication, Vol. 6, No. 4, 1999.
12. Rouvoy R., Beauvois M., Eliassen F. *Composing Components and Services using a Planning-based Adaptation Middleware*. In: 7th Int. Symp. on Software Composition (SC). LNCS 4954, Springer, 2008; 31-36.
13. Jiang S., Hallsteinsen S., Barone P., Mamelli A., Mehlhase S., Scholz U. *Hosting and Using Services with QoS Guarantee in Self-Adaptive Service Systems.* In: Eliassen, F., and R. Kapitza, R. (eds.): DAIS 2010, LNCS 6115, Springer, 2010; 15-28.
14. Hallsteinsen S., Jiang S., Sanders R. *Dynamic software product lines in service oriented computing*. In: 3rd Int. Work. on Dynamic Software Product Lines (DSPL), 2009.
15. Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.