Verifying Distributed Algorithms with Executable *Creol* Models

Wolfgang Leister
Norsk Regnesentral
Oslo, Norway
wolfgang.leister@nr.no

Joakim Bjørk and Rudolf Schlatte
Institute of Informatics, University of Oslo
Oslo, Norway
{joakimbj,rschlatte}@ifi.uio.no

Andreas Griesmayer
VERIMAG/UJF
Grenoble, France
andreas.griesmayer@imag.fr

***Abstract*—We show a way to evaluate functional properties of distributed algorithms by the example of the AODV algorithm in sensor networks, *Creol* models and component testing. We present a new method to structure the evaluation work into the categories of techniques, perspectives, arrangements, and properties using executable models. We demonstrate how to use this structure for network simulations and component testing in order to evaluate a large list of properties. We also show which properties are most suited to be evaluated by which technique, perspective, and arrangement.**

***Keywords*—formal analysis; modelling; model checking; testing; routing algorithms**

I. INTRODUCTION

With increasing miniaturisation, computational devices are becoming virtually omnipresent and pose new challenges in software development. We study arising questions on the example of wireless sensor networks (WSN) [1] consisting of spatially distributed autonomous sensor nodes that communicate using radio connections. Each node can sense, process, send and receive data. We concentrate on the verification of a *Distributed Algorithm* for ad-hoc networks between the sensor nodes to route data packets of the participating nodes. There are many requirements for WSN: routing must fulfil properties for quality of service (QoS), timing, delay, and network throughput; furthermore, we are interested in properties like mobility and resource consumption. Autonomous behaviour of the nodes leads to state space explosion during model checking, making evaluation a complex task that requires a combination of techniques from different verification approaches.

In this paper, which is based on a previously published report [2], we present a structured methodology to verification that introduces the categories of *techniques*, *perspectives*, *arrangements*, and *properties*. We combine this novel structure with techniques from simulation, testing, and model checking to create a new, unified method for verification of distributed systems. To this end, we use models in *Creol*, an object oriented modelling language that allows executable models. The novel idea behind our work is to employ one single executable model that is suitable for simulation, testing, and model checking, without the need to develop separate models for each task. We demonstrate the approach by evaluating a large set of properties on a network using the *Ad hoc On Demand Distance Vector* (AODV) routing algorithm [3].

The remainder of this paper is organised as follows: After briefly presenting the *Creol* language and related work, we discuss the AODV model developed previously (Section II), our categories for the validation process (Section III), present results from network simulation and component testing (Section IV), and conclude in Section V.

A. Executable Creol Models

Creol [4], [5] is an *object-oriented* modelling language, which provides an abstract, executable model of the implementation of components. The *Creol* tools are part of the *Credo* tool suite [6] that unifies several simulation and model checking tools. The *Credo* tools support integrated modelling of different aspects of highly re-configurable distributed systems, both structural changes of a network and changes in the components. The *Credo* tools offer formalisms, languages, and tools to describe properties of the model in different levels of detail; these formalisms include various types of automata, procedural, and object-oriented approaches.

To model components, *Creol* provides behavioural interfaces to specify inter-component communication. We use intra-component interfaces together with the behavioural interfaces to derive test specifications to check for conformance between the behavioural model and the *Creol* implementation. Types are separated from classes, and (behavioural) interfaces are used to type objects. *Creol* objects can have active behaviour and are concurrent, i.e., conceptually, each object encapsulates its own processor. During object creation a designated run method is automatically invoked. Asynchronous method calls, explicit synchronisation points, processor release, and under-specified, i.e., nondeterministic, local scheduling of the processes within an object provide flexible object interaction.

Creol includes a compiler, a type-checker, and a simulation platform based on Maude [7], which allow simulation, guided simulation, model testing, and model checking.

B. Related Work

Showing functional correctness and non-functional properties for algorithms employed for WSN helps the developers in their technical choices. Developers use a variety of tools, including measurements on real implementations, simulation, and model-checking. When developing algorithms for packet forwarding in a WSN, simulation results must be compared

 ISBN: 978-1-61208-132-8

with the behaviour of known algorithms to get a result approved [8]. Approaches using simulation, testing, and model checking during the development process use one or more of the following: modelling, traces, runtime monitoring by integrating checking software into the code (instrumentation) [9], or generating software from models automatically [10]. Simulation systems are used to analyse performance parameters of communication networks, such as latency, packet loss rate, network throughput, and other metrics. Most of these systems use discrete event simulation. Examples for such simulation systems include Opnet, OMNeT++, and ns-2 [11], or mathematical frameworks like MathWorks. Many of these tools have specialised libraries for certain properties, hardware, and network types.

The CMC model checker [9] has been applied on existing implementations of AODV by checking an invariant expressing the loop-freeness property. In that work, both specification and implementation errors were found and later corrected in more recent versions of both specification and implementations. CMC interfaces C-programs directly by replacing procedure calls with model-checker code, thus avoiding the need to model AODV. The model checking tools SPIN and *UPPAAL* have been used to verify properties for the correct operation of ad hoc routing protocols [12], such as the LUNAR and DSR algorithms [13]. Both LUNAR and DSR are related to AODV, but use different mechanisms. A timing analysis in *UPPAAL* uncovered that many AODV connections unnecessarily timed out before a route could be established in large networks [14]. Timed automata implemented in *UPPAAL* have been used for validating and tuning of temporal configuration parameters and QoS requirements [15] in network models that allow dynamic re-configurations of the network topology. The model checker *Vereofy* [16], part of the *Credo* tools, has recently been used to analyse aspects of sensor networks and AODV. We also used *Vereofy* as a reference for evaluating the properties and as source for the traces employed for the component testing.

The OGDC-algorithm used in certain sensor networks has been simulated and model-checked in Real-Time Maude [17]. The comparison of these simulation results in Real-Time Maude with simulation results in ns-2 have uncovered weaknesses in a concrete ns-2 simulation.

II. Modelling the Components and the Routing Algorithm

Distributed applications can be described in terms of components interacting in an open environment, based on the mechanisms of *Creol* [18]. This framework models components and the communication between these components, and executes the models in rewriting logic. Different communication patterns, communication properties, and a notion of time are supported. The lower communication layers use tight, loose, and wireless links.

Based on this work, we defined a model of AODV in a WSN using *Creol* [19] that expresses each node and the network as objects with an inner behaviour. The interfaces of the objects describe the communication between the nodes and

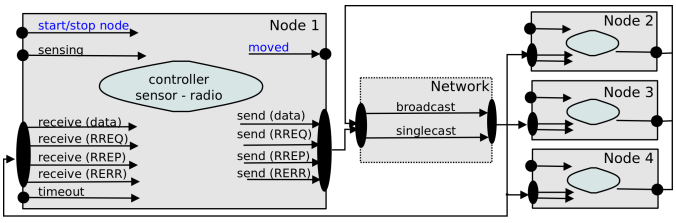

Figure 1. Objects of a WSN model and their communication interfaces.

the network object. In Figure 1, we show the object structure of the model, including the most important interfaces of a node. Within the nodes, its behaviour is implemented in *Creol* as routines that are not unlike real-world implementations.

The purpose of a routing algorithm is to establish a path between a source node and a sink node, so that data can flow from the source node to the sink node. AODV is a reactive routing protocol that builds up the entries in the dynamic routing tables of nodes only if needed. AODV can handle network dynamics, e.g., varying wireless link qualities, packet losses, and changing network topologies.

When a node wants to send a message to a sink node and the next hop cannot be retrieved from the local routing table, it initiates a route discovery procedure by broadcasting RREQ (route request) messages. Nodes that receive a RREQ message will either send a RREP (route reply) message to the node which originated the RREQ message if the route is known; otherwise the node will re-broadcast the RREQ message. This procedure continues until the RREQ message reaches a node that has a valid route to the destination node. The RREP message is unicast to the source node through multi-hop communications; as the RREP message propagates, all the intermediate nodes also establish routes to the destination. After the source node has received the RREP message, a route to the destination has been established, and data packages can be sent along this route.

The essential entries of the routing table include the next hop, a sequence number, and the hop count to the sink node. The latter is the most common metric for routing to choose between routes when multiple routes exist. The sequence number is a measure of the freshness of a route.

When communication failures imply a broken route, the node that is unable to forward a message will inform other nodes, so that the routing tables can be updated. To do this it sends a RERR (route error) message along the reverse route that is also stored in the nodes. Thus the source node will become aware of the broken route, and initiate a new route discovery procedure.

III. Methodology for Simulation, Component Testing, Model Checking

In the following section we show how to evaluate and validate the functional behaviour of the AODV model using *Creol* and the *Credo* methodology [6]. We present the *techniques*, *perspectives*, *arrangements*, and *properties* necessary

 ISBN: 978-1-61208-132-8

for the validation. We also show how to evaluate selected non-functional properties.

A. Techniques for Simulation, Testing, and Model Checking

In order to evaluate the properties of a model, several *techniques* are used to provide the necessary technical measures and procedures to make a model amenable to verification. In general, the following modifications can be applied to the model in preparation for simulation, testing, and model-checking:

Auxiliary variables are added to the model to improve the visibility of a model's behaviour. They must not alter the behaviour and are updated when certain relevant events happen, e.g., a counter is incremented when a new instance is created. When running a simulation these values can be extracted from the state information and visualised in an step-by-step execution, or after the *Creol* model terminates.

Assertions might be necessary depending on the functional requirements to check. While a number of properties can be checked at the final state using auxiliary variables, properties on the transient behaviour of the model require a check during runtime. For such cases, *Creol* provides *assertions* that stop the execution of a model when the condition is violated. The state that caused the violation of the property is then shown for further analysis.

Monitors are pieces of software that run in parallel to the actual model and are used for properties that go beyond simple assertions. A monitor constitutes an automaton that follows the behaviour of the model to decide the validity of a path.

Guarded execution replaces nondeterministic decisions by calls to a guarding object, the *DeuxExMachina*. This allows to check the behaviour of the model under different conditions, while still maintaining reproducibility of the runs. This technique also specifies certain parameters of the environment, like failure rates of the network.

Fault injection adds a misbehaving node (possibly after a certain time) to check error recovery properties. E.g., switch off a node when energy is used up, or inject other errors. This can be implemented by sub-classing nodes and implementing certain misbehaving routines in the subclass.

Property search employs model checking techniques to check whether certain conditions hold for all or a given subset of states. Currently, property search must be specified in *Maude* for a *Creol* model.

B. Perspectives

A *perspective* describes the scope of an evaluation. For the AODV model we developed two perspectives: (*a*) observing the behaviour of the entire network configuration including all nodes and the network; (*b*) observing the behaviour of one node. Testing, simulation, and model checking can be performed from different perspectives and levels of detail for a given model. For AODV, a holistic perspective focuses on the networking aspect of the nodes, implementing all the involved nodes and the network in one model. However, for model checking, such a model leads to a high number of states, and long execution time. Therefore, for realistic models the networking perspective is not feasible.

For the perspective of testing a single node we use the same model code for the nodes in the holistic perspective, but instantiate only one node explicitly. The network is replaced by a *test harness* that impersonates the network and the remaining nodes. The behaviour and responses of the test harness are determined by a rule set that is derived from traced messages between the nodes, as outlined in Section IV-B.

C. Arrangements

An *arrangement* denotes a set of configuration settings that influences how the model operates. Examples for arrangement entities that can be selected in the *Creol* models, together with implementation details for the AODV model are given in the following:

The *communication behaviour* in our model can be set to be either reliable, non-deterministic, or one of several packet loss patterns. Note that non-deterministic behaviour in a simulation currently is not useful due to restrictions in the implementation of the underlying run-time system. Using the differences in communication behaviour, we can study how the algorithm behaves when communication packets can get lost.

Topology changes are used to check the robustness of the protocol. They can be triggered by certain events, e.g., after a certain amount of messages, or after a certain amount of time for timed models. A topology change affects the connection matrix in the network and triggers the AODV algorithm to find new routes in the model.

The *timed model* is realized using discrete time steps and introducing a global clock in the network object and internal clocks in the nodes, which are synchronised when a task is performed in one or more nodes. This allows, e.g., to reason about messages being sent simultaneously, which eventually will lead to packet loss. Also the effect of collisions can be shown without using non-deterministic packet loss. The use of a timed model is most viable together with topology changes since the topology needs to be re-installed for a state when another branch is searched in model checking.

Energy consumption is modelled using an auxiliary variable in each sensor node with an initial amount of energy. For each operation a certain amount of energy is subtracted until the capacity is too low too perform operations on the radio, indicating a malfunction of the sensor node. Such a node does not perform any actions and represents a topology change of the network, since given path are no longer valid. This allows us to identify in which cases an energy-restricted network can perform communication and whether AODV can find routes around an energy-empty node.

Note that arrangements for memory and buffer sizes can be implemented similarly. When maximum memory size is reached, a node will stop to perform certain actions.

Timeouts are modelled nondeterministically by use of a global guarding object and can occur between a message is sent and the corresponding answer is received. AODV employs time-outs in order to work in environments where communication

 ISBN: 978-1-61208-132-8

errors can occur and sends messages repeatedly in case an expected reply has not been received from the network.

D. Properties

A *functional property* is a concrete condition that can be checked for given arrangements, while non-functional properties are values given by metrics. For AODV, we chose the following functional properties: correct-operation, loop-freeness, single-sensor challenge-response properties, shortest-path, deadlock-freeness (both, for node, and for protocol), miscellaneous composed system properties, and non-functional properties.

Correct-operation: For a routing algorithm to be correct, it must find a path if a path exists, i.e., it is *valid* for some duration longer than what is required to set up a route from sender to receiver [12], [13]. Checking this property requires the algorithm-independent predicate whether a route exists. In the absence of topology changes, this predicate can be calculated beforehand. When topology changes are possible, however, we need to check the existence of a path between sender and receiver at any step in the algorithm. Since checking this property in *Creol* involves explicitly visiting all nodes, this increases the reachable state-space of the model. To evaluate this predicate effectively, a suitable implementation would be to interface a *Maude* function. Unfortunately, this is currently not supported by *Creol*.

A related property to evaluate is whether a route is re-established after a transmission error, given a path still exists. We also evaluate how long the path is interrupted after a transmission error occurs.

Loop-freeness: A routing loop is a situation where the entries in the routing tables form a circular path, thus preventing packets from reaching the destination. The invariant for loop-freeness [9] of AODV must be valid for all nodes. It uses *sequence numbers* of adjacent nodes, and the number of hops in the routing tables as input. The loop-freeness property is checked every time a message is transmitted between nodes. To do this, the network-object calls a routine that checks the loop-freeness invariant in an assertion. Since this assertion is complex, and contains nested loops, it should be implemented as a call to a *Maude* function instead of *Creol* code.

Single-sensor challenge-response: The reaction of one node under test is checked using component testing (Section IV-B). Messages are sent to the node under test, and the responses from this node are matched against all correct responses. The correct responses are extracted from specifications or from running simulations using different implementations. The single-sensor properties that can be checked express a certain behaviour of the absence of a certain behaviour after a challenge, e.g., whether an incoming message leads to a specified state change in the node, or whether the node sends an expected response messages.

Shortest-path: Here, we investigate whether the AODV algorithm finds the shortest path for the paths between source and sink node; also other metrics for paths could be checked. While AODV finds the shortest path in the case of no packet loss, it does not always fulfil the shortest-path property in the case of packet loss. To check this property we count the number of hops that each payload-message takes from the source to the sink and compare it with the shortest existing path between source and sink.

Deadlock freedom: Deadlocks in a node, in the protocol or in the model are a threat to robustness, and can reveal errors in the specification, implementation, or model. Deadlocks will make the model execution stop in an error state.

Miscellaneous composed-system properties: Examples are properties that state whether valid routes stay valid, avoidance of useless RREQ messages, number of messages received, timing properties and network connectivity. Most of these are implemented by adding counter variables, and predicates.

Non-Functional Properties: Non-functional properties from the application domain, such as timing, throughput, delivery ratio, network connectivity, energy consumption, memory and buffer sizes, properties of the wireless channel, interferences, mobility, or other QoS properties, can be evaluated by using counter variables and additional *Creol* code.

IV. Holistic and Component Testing

An instrumented *Creol* model can be used for the different verification and testing techniques in the *Credo* methodology: symbolic simulation, guarded test case execution, and model checking. Currently, the auxiliary variables for assertions and the state of the monitors need to be added as *Creol* code that is executed together with the model code. This increases the size of the states and therefore poses a handicap for model checking.

A. Holistic Testing

For our evaluation of the network properties we used simulation using mainly techniques such as auxiliary variables, and assertions. Most of our experiments used a network with symmetrical communication via four sensor nodes and one sink node. We simulated the AODV model using various arrangements using reliable networks, lossy networks, timeouts, energy consumption, and timed modelling. We also checked selected properties from Section III-D.

Reliable communication: As long as the network is connected, the evaluations showed that the modelled AODV algorithm fulfils the properties from Section III-D. We emphasised on the evaluation of packet loss, and loop-freeness assertion. Other predicates for loop-freeness were also used (which failed as expected), and small, faulty changes in the model were introduced (which led to expected failures of the loop-freeness property). The shortest path property was fulfilled in all simulated occasions.

Lossy communication: When simulating lossy communication, both for singlecast, and for broadcast messages the packet loss rate increases as expected. We also observed an increased number of RREQ and RREP messages in the system, using auxiliary variables.

Timeouts: The model allows re-sending of lost RREQ messages up to a certain number of times, using a timeout

 ISBN: 978-1-61208-132-8

TABLE I
NUMBER OF REWRITES AND RUN-TIME FOR SAMPLE ARRANGEMENTS AND PROPERTIES.

#	t steps	energy	loss	timeout	#rewrites	time
	500	–	none	never	$9.4 \cdot 10^6$	17.1s
	5000	–	none	never	$62.8 \cdot 10^6$	114.8s
	500	–	10%	never	$10.7 \cdot 10^6$	19.5s
5	500	–	10%	1/10	$12.1 \cdot 10^6$	22.3s
	500	50	10%	1/10	$8.3 \cdot 10^6$	15.5s
	untimed	50	10%	1/10	$11.6 \cdot 10^6$	17.9s
	untimed	–	10%	never	$32.5 \cdot 10^6$	14.8s
6	untimed	–	10%	never	$90.5 \cdot 10^6$	40.9s
15	5000	–	none	never	$2.7 \cdot 10^9$	31m
30	5000	–	none	never	$24.8 \cdot 10^9$	8h

mechanism. We could observe that this mechanism decreased the packet loss rate, but at the same time does not prevent all packet loss for payload packets.

Energy consumption: Using the energy consumption arrangement we can force a communication failure of certain nodes after some actions. Using this arrangement we can study the re-routing behaviour in detail, including the packet loss rate.

Timed model: Using the timed model we can study the number of time steps needed for sending messages, as well as controlling the number of actions being performed simultaneously. We observed that the packet loss rate is different to the untimed case, which is expected.

Using the timed model, we could observe a model deadlock, which is caused by the way the model is implemented, and certain properties of the current implementation of the *Creol* runtime system. This observation made changes in the model implementation necessary using asynchronous method calls.

The developed *Creol* model was evaluated by using simulation for sample arrangements and properties. The entire model contains about 1600 lines of *Creol* code, excluding comments. After compilation, the resulting code size was about 1050 lines of Maude code, depending on the arrangement. We varied the timing behaviour, the energy consumption, the message loss behaviour, and the timeout behaviour of the model, as well as the number of nodes. The results for the tested cases considering the number of rewrites, and execution time on an AMD Athlon 64 Dual core processor with 1.8 GHz is shown in TABLE I. The timing behaviour, and the number of nodes are the most significant parameters.

While these values may sound high for a simulation system, we emphasise that the purpose of the *Creol* model is to offer one model that is suitable for several perspectives. While the transition from simulation to model checking consists in changing some few Maude statements, the search space during model checking gets combinatorially too high to be viable, already for a low number of nodes.

B. Component Testing of One Node

For component testing, we use one node under test with the same code as for holistic testing. However, we replace the network and all the other nodes using a test harness, as shown in Figure 2. The test is then performed by studying the output messages of a node when given input messages are applied.

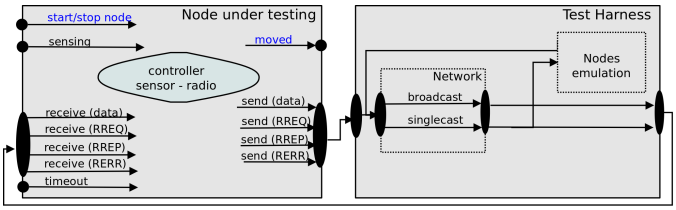

Figure 2. Testing of one node using the network object as a harness.

1) Test harness: The task of the test harness is to send messages to the interfaces of the node under test, and to observe its answers. Both input messages and expected answers can be generated from the specification or from traces of real systems or other simulations.

Although incoming broadcast, singlecast and outgoing packets involve invoking different methods, the *Creol* language, with its object-level parallelism, makes it easy to encode a test case as a single sequential list of statements. Incoming messages are stored in a one-element buffer; the test case simply performs a blocking read on that buffer when waiting for a message from the object under test, before sending out the next message to the object. In that way, both creating a test case by hand and generating test cases from recorded traces become feasible.

A test verdict is reached by running the test harness in parallel with the object under test. If the test harness *deadlocks*, it expects a message from the object under test that is not arriving – in that case, a test verdict of *Fail* is reached. The other reason for test failure is an incoming message that does not conform to the expectations of the test harness; e.g. by being of the wrong type or having the wrong content.

A test verdict of *Success* is reached if the test harness completes the test case and the object under test conforms to the tester's expectations in all cases.

2) Traces: In addition to domain-specific single-object properties test cases can be generated from model implemented with *Vereofy* [16]. To receive the traces from *Vereofy* the content of all variables within the nodes and buffers in the network, before and after each step, and the exchanged data are collected. When the state information is removed we receive a sequence of messages that are exchanged simultaneously.

Traces received from the node under test are tested against *message patterns*, i.e., we remove details that could lead to spurious test failures not expressing a malfunctioning system. For example, the message sequence number can be chosen by the node, the only requirement is that it be monotonically increasing. This property is checked using an invariant in the tester, but a different concrete message number than that used by the *Vereofy* model cannot lead to test failure.

 ISBN: 978-1-61208-132-8

V. Conclusion

We presented the evaluation of a *Creol* model of AODV. We introduced the dimensions of *techniques*, *perspectives*, *arrangements*, and *properties* for this evaluation. We divided the properties used for this evaluation into six property classes, and performed network simulations of the composed system, and component testing of a single node.

Using the network simulation, we evaluated several arrangements. While most most of the properties were fulfilled as expected, some properties did not validate in the simulation, either due to bugs in the model, properties of the modelled AODV algorithm, artificially introduced bugs in the model, or property variants that are not supposed to validate successfully. In one occasion, we could detect deadlocks in the model in a timed-model arrangement, which could be recognised and fixed afterwards. Evaluating other protocols, such as the proactive dynamic routing protocol, is possible, but requires a new model.

Using component testing, we validated the correct behaviour of a single node against properties extracted from the specification of the AODV algorithm. No deviations from specified component behaviour were identified in this process, which is unsurprising since components had already been extensively used for simulation and animation during initial model development at that point in time. However, the test suite served as an excellent help in regression testing during subsequent changes and extensions of the model.

Evaluating the properties of the AODV algorithm, we encountered several challenges, such as modelling suitable abstractions, using language constructs of *Creol*, and observing the properties from a suitable perspective. The major challenge when evaluating the AODV algorithm from a network perspective is to avoid a high number of states in the underlying interpreter. The use of *Creol functions* that can be excluded from the state in model checking would be desirable. This feature is, however, not yet available in the current *Creol* tools.

We found the *Creol* language and the tools useful in the evaluation of the AODV algorithm, and to gain insight in how complex algorithms like AODV work. We observed how small changes in the algorithm, and in the chosen arrangement, imply changes in its behaviour. We also detected the breach of certain properties, which will lead to further investigation of the reasons, removal of this misbehaviour, and, eventually, to a better understanding of AODV, and the algorithms used for sensor networks.

Acknowledgements

This research is part of the EU project IST-33826 CREDO: *Modeling and analysis of evolutionary structures for distributed services*. The authors want to express their thanks to their colleagues involved in the *Credo* project for their support during this work, especially Sascha Klüppelholz, Joachim Klein, Immo Grabe, Bjarte M. Østvold, Xuedong Liang, Marcel Kyas, Martin Steffen, and Trenton Schulz.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] W. Leister, J. Bjørk, R. Schlatte, and A. Griesmayer, "Validation of creol models for routing algorithms in wireless sensor networks," Norsk Regnesentral, Oslo, Norway, Report 1024, 2010.

[3] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561 (Experimental), Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[4] E. B. Johnsen and O. Owe, "An asynchronous communication model for distributed concurrent objects," *Software and Systems Modeling*, vol. 6, no. 1, pp. 35–58, 2007.

[5] M. Kyas, *Creol Tools User Guide*, 0.0n ed., Institutt for Informatikk, Universitetet i Oslo, Postboks 1080 Blindern, 0316 Oslo, Norway, May 2009.

[6] I. Grabe, M. M. Jaghoori, B. Aichernig, T. Blechmann, F. de Boer, A. Griesmayer, E. B. Johnsen, J. Klein, S. Klüppelholz, M. Kyas, W. Leister, R. Schlatte, A. Stam, M. Steffen, S. Tschirner, X. Liang, and W. Yi, "Credo methodology. Modeling and analyzing a peer-to-peer system in Credo," in *3rd International Workshop on Harnessing Theories for Tool Support in Software*, 2009.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada, "Maude: Specification and programming in rewriting logic," *Theoretical Computer Science*, 2001.

[8] I. Stojmenovic, "Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions," *IEEE Communications Magazine*, vol. 46, no. 12, pp. 102–107, 2008.

[9] M. Musuvathi, D. Y. W. Park, A. Chou, D. R. Engler, and D. L. Dill, "CMC: a pragmatic approach to model checking real code," in *OSDI, Usenix*, 2002.

[10] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri, "A framework for modeling, simulation and automatic code generation of sensor network application," in *Proc. Fifth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON 2008, June 16-20, 2008, San Francisco, USA*. IEEE, 2008, pp. 515–522.

[11] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–10.

[12] O. Wibling, J. Parrow, and A. N. Pears, "Automatized verification of ad hoc routing protocols," in *FORTE*, ser. Lecture Notes in Computer Science, vol. 3235. Springer, 2004, pp. 343–358.

[13] ——, "Ad hoc routing protocol verification through broadcast abstraction," in *FORTE*, ser. Lecture Notes in Computer Science, vol. 3731. Springer, 2005, pp. 128–142.

[14] S. Chiyangwa and M. Z. Kwiatkowska, "A timing analysis of AODV," in *FMOODS*, ser. Lecture Notes in Computer Science, vol. 3535. Springer, 2005, pp. 306–321.

[15] S. Tschirner, X. Liang, and W. Yi, "Model-based validation of QoS properties of biomedical sensor networks," in *EMSOFT '08: Proceedings of the 7th ACM international conference on Embedded software*. New York, NY, USA: ACM, 2008, pp. 69–78.

[16] C. Baier, T. Blechmann, J. Klein, S. Klüppelholz, and W. Leister, "Design and verification of systems with exogeneous coordination using Vereofy," in *Proc. 4th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2010), Part II*, ser. LNCS, vol. 6416. Springer-Verlag, Oct. 2010, pp. 97–111.

[17] P. C. Ölveczky and S. Thorvaldsen, "Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude," in *FMOODS*, ser. Lecture Notes in Computer Science, vol. 4468. Springer, 2007, pp. 122–140.

[18] E. B. Johnsen, O. Owe, J. Bjørk, and M. Kyas, "An object-oriented component model for heterogeneous nets," in *FMCO*, ser. Lecture Notes in Computer Science, vol. 5382. Springer, 2007, pp. 257–279.

[19] W. Leister, X. Liang, S. Klüppelholz, J. Klein, O. Owe, F. Kazemeyni, J. Bjørk, and B. M. Østvold, "Modelling of biomedical sensor networks using the Creol tools," Norsk Regnesentral, Oslo, Norway, Report 1022, 2009.

 ISBN: 978-1-61208-132-8