Project Number: 33826

CREDO

Modeling and Analysis of Evolutionary Structures for Distributed Services

Deliverable D6.4
Validation

Due Date:	19-02-2010
Submission Date:	19-02-2010
Resubmission Date:	12-04-2010
Start date of project:	01-09-2006
Duration:	3 years
Extension:	4 months
Lead Participant:	ALMENDE

Project funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level: PU Public

Project Participants

Role	No	Name	Acronym	Country
CO	1	Stichting Centrum voor Wiskunde en Informatica	CWI	NL
CR	2	Universitetet i Oslo	UIO	N
CR	3	Christian-Albrechts-Universität zu Kiel	CAU	DE
CR	4	Dresden University of Technology	TUD	DE
CR	5	Uppsala Universitet	UU	S
CR	6	United Nations University, International Institute for Software and Technology	UNU-IIST	JP
CR	7	Almende B. V.	ALMENDE	NL
CR	8	Rikshospitalet - Radiumhospitalet HF	RRHF	N
CR	9	Norsk Regnesentral	NR	N

C0 = Coordinator CR = Contractor
NL = Netherlands N = Norway
DE = Germany S = Sweden
JP = Japan

Contents

1 Introduction

This deliverable presents the results of the validation phase of the *Credo* project for the two case studies ASK and BSN. Starting point for this validation is the set of final models for both case studies presented earlier in Deliverable D6.3. Throughout the validation phase, three different tool sets have been used, each with a different modeling language in scope:

- The *Eclipse Coordination Tools (ECT)* and *Vereofy* for simulating and validating Reo circuits and (Constraint) Automata models
- The *Creol (CreolE) Tool Set and Eclipse Plugin* and the *Creol Testing Tools* for simulating and validating *Creol* models
- The *UPPAAL Tool* and *VerifyTA* for simulating and validating Timed Automata models

Outline

Sections 2 and 3 of this deliverable are devoted to the ASK case study and the BSN case study, respectively. In each section, we start with a brief overview of the case study, followed by the validation results for each of the tool sets mentioned above. We conclude each section with a quantitative evaluation providing statistics about the validated properties and performance of the tools, an overview of the original requirements on the *Credo* modeling language and tools, an indication if and how these original requirements are met, and lessons learnt throughout the validation phase of the project.

Live-CD

Deliverable D6.4 and its annexes can be found on the latest version of the *Credo* Live-CD. The Live-CD also contains all other project deliverables, the (updated) final models, and the tools and documentation necessary to view, simulate, and validate the models.[1]

[1] Because of licensing restrictions, the *UPPAAL* and *VerifyTA* tools are not available on the *Credo* Live-CD.

2 Case study 1: The ASK System

Within the ASK case study, the validation phase serves three purposes. First, by validation we are able to validate, correct and improve the final models of Deliverable D6.3. Second, validation allows us to assess properties by which we can improve the implementation of the ASK system. Improving the actual software of ASK is beyond the scope of the *Credo* project, but we will explain throughout the discussion of the validation results how we think we can use the validation results in the future for this purpose. Third, by using the *Credo* validation tools, we directly assess the quality and usefulness of these tools.

2.1 Overview of the ASK Case Study

In the ASK Case Study, we created several types of models of various parts of the ASK system (For a detailed overview of the ASK System itself, see Deliverable D6.3 Section 2). We splitted the final modeling effort for the case study into three "subprojects": *ReASK*, *CreASK* and *tASK*.

In the *ReASK* project, which builds upon the initial modeling presented earlier in Deliverable D6.2, we have focused on modeling the ASK system in terms of a REO network at various levels of abstraction. At the lowest level of abstraction, we modeled the components in terms of automata, which resemble the initial model of ASK. In the validation phase, the REO networks (circuits) and automata have been validated with the *ECT* tools and the *Vereofy* tool. The results are presented in Section 2.2.1.

In the *CreASK* project, we have focused on a specific part of the ASK core system, namely the *thread-pools*, which are present in each of the components of the system. For this part, we created *Creol* models to analyze functional properties. We performed several simulations and validations of these models with the *Creol* tools, the *Creol* eclipse plugin and the *Creol* testing tool set. An overview of the validation results is given in Section 2.2.2.

Finally, in the *tASK* project, we have used the Creol models of the thread-pools as a basis for timed automata models. We used these models to assess the schedulability of a particular amount of tasks with strict deadlines and inter-arrival times, given a thread-pool with certain dimensions, by using the *UPPAAL* and *VerifyTA* tools. Section 2.2.3 is devoted to the results of this validation.

2.2 Validation of the Final Models

We discuss the validation for the models of the three projects *ReASK*, *CreASK* and *tASK* in three separate sections.

2.2.1 *ReASK*: Validation of the REO Models of ASK

For the ASK case study, a model covering the control and message flow of the ASK system with its hierarchical layers has been developed. It consists of automata for each component on the lowest layer, and Reo circuits composing the upper thread, process, system, and context layers (see Deliverable D6.3 and Annex D6.3.2). The model is parameterized in many ways, e.g., queue sizes and the number of monks used within the abbeys of ASK. The version we mainly dealt with in the model checking phase consists of 328 component instances and channels (63 different types) from which 73 constitute the scheduler process, 38 the matcher process, 40 the reception process, and 141 the resource manager process. The remaining channels form the coordinating network. The full Vereofy representation of our model in RSL and CARML (see Deliverable D5.3 and Annex D5.3.1) uses 393 boolean variables to encode the state space of the composite system. The number of reachable states of the processes which we could build in full detail reaches a magnitude of up to 10^{10} states. The total number of dataflow locations was 1007 and for encoding of the data values at each location 11 boolean variables (for 290 distinct message values) were used.

Five sub-components interacting with each other form the ASK contact engine. The complex communication between the reception, matcher, executer, scheduler, and resource manager as well as their coordination is completely represented in the Vereofy model. This inherent complexity of the model, together with the heavy dependency on the data values of the transferred messages, made formal verification a challenging task. Subsequent optimizations of the model checker provided significant improvements in the ability to handle such complex systems, allowing the handling in full detail up to the process layer.

We were able to use Vereofy to verify (and falsify) various properties and find different kinds of errors in the model, like wrongly ordered messages, deadlocks, missing robustness features, incorrectness of assumptions for simplifications, etc., which could then be analyzed with the help of the counterexamples created by Vereofy and fixed in the model. For components on the process, system, context level we did a pre-computation of a good BDD (Binary Decision Diagram) variable ordering to start the model checking with a compact representation of the model. These computations took only minutes for most of the components. Only for a few complex components we had to spend some hours on this. With this pre-computation it was possible to compose systems within only a few seconds and perform model checking within minutes.

While model checking of the fully detailed model for the two (most complex) top layers was currently out of reach due to the complexity of the model, it was possible to use Vereofy to simulate the whole model by generating random execution traces with stepwise, iterative building (Deliverable D5.6).

The key techniques to deal with the complexity of system and context layer within the model checker were abstraction of the data domain, replacing component specifications by simpler ones with equivalent I/O-behaviour or generic components with non-deterministic behavior, and techniques for creating better variable orderings supported by the latest version of Vereofy. Applying them to the ASK system allowed us to check some safety properties as well as the quality and robustness of the underlying coordination and communication structures and mechanisms for the contact engine of ASK.

In the following we give some examples of results in the model checking phase for the ASK case study. A detailed list including all relevant properties checked for the ASK case study can be found in Annex D6.4.5.

- *Missing task creation in the initial phase of the Scheduler Monk:* while checking the correctness of the order of incoming tasks and outgoing requests we found that the main process of the scheduler never created the necessary initial tasks to produce the necessary initial request.

- *Robustness of monks:* we used lossy filter channels for routing incoming tasks of a monk to its responsible sub-process. At the same time, a token was consumed from a buffer (finished_buffer), which should be restored once the sub-process finished working on the task, thus allowing a new task to be accepted. When unknown task types arrived, the token was consumed and the task rejected, i.e. lost. The token was thus never restored by any of the sub-processes, as none of them became active. The monk was not robust for faulty task messages. The fix replaced the lossy filters by blocking variants and the standard Reo nodes by route nodes to ensure exclusive task forwarding. After applying the fix onto all monk structures, we were able to remove another structural problem in the reception and matcher monk which was a consequence of replacing the node by a route node.

- *Simplification of monks:* in the final model we had a finished buffer for each sub-process in the monks. Assuming that only one process should be active at time, there would be no need for more than one buffer and thus a simpler monk implementation having only one buffer cell would still be sufficient. We first showed that our assumption is correct, i.e., that it was never the case that the buffers stored more than one token. After replacing the monks by simpler ones we were able to show bisimilarity for the original and simplified monk.

- *Errors in the hashtable implementation for Connectoid Warehouse:* the hashtable specification was given in terms of an automaton, which was then one-to-one translated into a CARML module. Due to the very regular structure of the automaton, the modeling involved a lot of copy-and-paste operations. We found all mistakes caused by wrong copy-and-paste operations, by implementing another module having the hashtable functionality and checking for bisimilarity of the two implementations and showing that both model the hashtable specification properties.

All models, verified properties and deliverables can be found on the *Credo* Live-CD.

2.2.2 *CreASK*: Validation of Creol Models of ASK Thread-Pools

For the modeling activity using the Creol language, it was decided to create a behavioral model of the *thread pool* (abbey) functionality contained in the ASK system (see Deliverable D6.3 and Annex D6.3.1). This particular component was chosen mainly for three reasons. First, different parts of the ASK system contain their own implementations of thread pools, with similar but slightly different semantics. It was expected that the modeling activity would help in finding, enumerating and understanding the various thread pool implementations in the existing code base. Second, the thread pool component possesses a clean interface, with well-defined functionality and communication patterns with the rest of the system. Third, the model was expected to be suitably different to the BSN case study so as to yield different results, and be amenable to different analysis methods.

All these assumptions proved to be correct; various thread pools were successfully modelled in Creol and used by other work packages to good effect. Low-level and high-level (abstract) models of constant-sized (dabbey) and self-balancing (sabbey) thread pools were used to simulate the system's behavior, generate test inputs, and validate the ASK system's behavior.

Simulation was used during the creation of the models to gain understanding about language semantics and available tools, and to validate the correctness of the model's behavior. As such, simulation was done in an ad-hoc way, more for demonstration purposes than for generation of further results.

The Creol models of the ASK thread pools were extensively used in Work Package 5 in the *Testing* activity, which resulted in a number of methods and tools (and publications as well). The two most important approaches are *concolic execution* and *passive trace-based testing*.

- We developed a way of using dynamic symbolic execution (also known as concolic execution) to generate an exhaustive set of test inputs from a specification. The approach was implemented on top of the Creol interpreter using the YICES SMT solver and validated against the thread pool model. This approach is described in Deliverable D5.5.

- To generate test cases and test the behavior of the existing ASK system against its Creol model, we developed a method for *passive trace-based testing*. In brief, the behavior of the ASK system is logged (using aspect-oriented programming), the resulting logs are converted to Creol models of the real system's behavior, and the modeled behavior is replayed using the Creol model as a test oracle. The approach is described in Deliverables D5.2 and D5.5. The test case generator tool implementing the generation and execution of test cases using this method was implemented as part of the Creol plug-in for the Eclipse integrated development environment.

All models, validation results and deliverables can be found on the *Credo* Live-CD.

2.2.3 *tASK*: Validation of UPPAAL Models of ASK Thread-Pools

For the schedulability analysis of the ASK system, we modeled the *dabbey* variant of the ASK thread pools. This variant has a fixed number of monks (threads) and a fixed-size task buffer. The details of the UPPAAL models corresponding to the dabbey can be found in Annex D6.4.4. The Dabbey is modeled in a flexible way such that by adjusting the following parameters, one can experiment with different settings of the abbey:

- The first thing one can change is the number of monks for the modeled abbey. By increasing the number of parallel monks, one can perform tasks faster. This means that smaller deadlines and inter-arrival times can be used.
- In order to perform experiments, the deadline and inter-arrival values can be changed in order to get the right value for schedulability.
- Furthermore, one can change the number of different task types.

In order to compare the schedulability of tasks for different numbers of monks and different inter-arrival times, we created a script to automatically invoke the schedulability analysis for different parameter values using the UPPAAL command-line verifier. The output of the script can be plotted in a 3D plane. An example is given in Figure 1. In this example, the number of different tasks is 9, while their computation times are defined as $\{8, 9, 9, 10, 10, 10, 11, 11, 12\}$. On the Z-axis of the plot, the minimum possible deadline for the tasks to be schedulable can be read, for different amounts of monks in the abbey and different inter-arrival times. Several conclusions can be drawn from this diagram, like:

- For the given set of task types, abbeys with size 6, 7, 8 and 9 perform equally well.
- For the given set of task types, if we allow less strict deadlines for the tasks, the minimum inter arrival times for abbeys with size 1, 2 or 5 can be improved.
- For the given set of task types, an abbey with 2 monks (minimum IAT 5, corresponding minimum deadline 13) performs more than twice as good as an abbey with 1 monk (minimum IAT 10, corresponding minimum deadline 14).

Such interesting results can be applied by Almende in the near future to improve the performance of the abbeys in the ASK system.

All models, validation outputs (plots) and deliverables can be found on the *Credo* Live-CD.

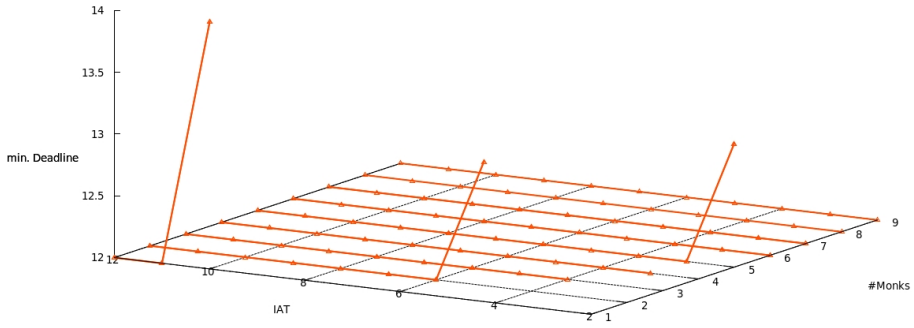

Figure 1: Schedulability analysis results for Dabbey with different numbers of monks and inter-arrival times.

2.3 Assessment of Initial Requirements

Overview of Assessment Scenarios

As we formulated in the addendum of Deliverable D6.1, our intentions with the ASK Case Study were twofold: to use it as an assessment of the project results, their quality and applicability in a real-life industrial setting, and to provide with it a basis for exploitation of the project results during the future development and maintenance of the ASK system. At that time of writing, Almende recognized three future developments on ASK relevant in the context of the *Credo* project: *scalability* of ASK, *integration* of multiple ASK instances, and *personalization* of ASK, of which the focus in *Credo* would be on the improvement of the *scalability* of ASK. We quote from the addendum to D6.1: "The focus of the "ASK CS" case study is on modeling, analyzing and implementing these scalability issues." At the beginning of the project, we identified four scenarios:

- **SC.1**: better exploitation of meta-information within ASK
- **SC.2**: optimization of local task scheduling strategies
- **SC.3**: optimized distribution of the ASK components
- **SC.4**: distributed replication of the ASK components

Throughout the project, these initial scenarios have been partially addressed, partially replaced by other scenarios, as follows. Scenario **SC.1** was the subject of the initial modeling. Scenario **SC.2** has been addressed in the final modeling and validation: subproject *tASK*, using UPPAAL and timed automata. Scenario **SC.3** has been indirectly addressed in subproject *reASK*, which built upon the insights gained throughout the initial modeling effort on *SC.1*, but extended the modeling effort for the small first scenario to a holistic modeling effort for the entire core of the ASK system. Finally, **SC.4** was replaced, prior to the start of the final modeling phase, by a scenario focusing on the thread-pools in the ASK System (subproject *CreASK*). This was done because the latter scenario was expected to better cover the set of tools developed in the case study.

Overview of Initial Requirements

In the Addendum on Deliverable D6.1, we formulated the following requirement categories:

1. *Credo* should support the modeling of ASK *structure*, ASK *behavior*, *location* and *time*;
2. *Credo* should support the modeling of constraints on *time*, *memory size* and *network bandwidth*;
3. It must be possible to verify *functional* properties of a reconfigurable system based on a *Credo* model created for it;
4. It must be possible to verify *non-functional* properties of a reconfigurable system based on a *Credo* model created for it;
5. The *Credo* tools must be applicable to individual components as well as compositions of components;
6. The *Credo* tools must provide ways to (semi-)automatically create models based on source code of the ASK system;
7. The *Credo* tools must be able to verify non-functional properties at runtime;
8. The *Credo* tools must provide information about functional and non-functional properties in an attractive visual manner;

Categories 1, 2, 5, 6 and 8 were covered by the final modeling. The assessment of requirement categories 3, 4 and 7 is considered to be part of this validation deliverable. However, for this document to be self-contained, we repeat the conclusions with regard to the requirement categories covered during the final modeling.

Assessment of the Results compared with the Initial Expectations

RC1. As we already concluded in our report on the Initial Modeling (Deliverable D6.2), *Credo* supports the modeling of structural and behavioral aspects of software systems, as well as of timing aspects. Location as such can *not* be explicitly represented in any of the *Credo* modeling languages.

RC2. Constraints on time can be modeled and analyzed in the UPPAAL tool. However, the *Credo* tools are less suited for modeling constraints on memory size or network bandwidth.

RC3. Functional properties, especially those mentioned as need to have in the addendum of Deliverable D6.1 (i.e., system provides the same functionality in different distributions, changing internals of a component has impact on its interface, a request cannot disappear in a certain system configuration) can be assessed with the *Credo* tools both for the networks (Reo, Vereofy, property verification) as well as for the components (Creol, Creol tools, simulation and testing techniques). We consider the multitude of ways to verify functional properties as a very strong aspect of the *Credo* tool suite. The *Credo* methodology provides some (but not yet sufficient) insight into which way of modeling and verification should be used for which kind of functional property.

RC4. A large part of the non-functional properties mentioned as need to have in the addendum of Deliverable D6.1 (i.e., a task can(not) be performed in a certain amount of time, an amount of tasks can(not) be performed within a time window, a request can(not) be handled in a certain amount of time, an amount of requests can(not) be handled within a time window) can indeed be verified with the *Credo* tools. The UPPAAL and VerifyTA tool and timed automata are very well suited for the verification of timing properties like the above. The *schedulability analysis technique*, developed in the context of the *Credo* project, covers precisely the aspects necessary for reasoning about the most important non-functionals for ASK. Other properties, dealing with memory or network bandwidth, are *not* addressed by the *Credo* tools.

RC5. The compositionality of especially REO makes the *Credo* language applicable to individual components as well as component compositions.

RC6. *Credo* definitely does *not* support (semi-)automatic creation of models based on source code of ASK, a requirement we considered as a necessity for broader adoption of the *Credo* tool suite, but which proved to be infeasible within the resource and time limits of the *Credo* project.

RC7. Given the regular execution times for property verification, this nice to have requirement is not yet met by the *Credo* tools.

RC8. Finally, the attractive visual manner in which models can be represented was not per se within scope.

2.4 Quantitative Evaluation

Table 1 shows statistics for the RSL/CARML model of the ASK system, as used by the Vereofy model checker, giving information about the different parts and levels of the ASK system model that were verified.

The first column of the table shows the number of boolean BDD variables used to encode the system. The next column shows the time spent to calculate a suitable variable ordering for the system. After such a variable ordering is calculated, it can be reused, with the next column showing the time spent to build a symbolic representation of the system using the previously calculated variable ordering. The next column shows the number of BDD nodes used for the symbolic representation of the transition function (generally the most complex part) of the system. For the systems at the Thread level of the ASK model, the default variable ordering generated by Vereofy already yields an efficient symbolic representation, so that the separate step of generating an optimized variable ordering was unnecessary and could be skipped.

The next three columns then show the number of temporal logic properties (see Appendix D6.4.5 for details) checked for the given systems and the average and maximum runtime of the model checker used to verify or falsify the properties.

For some of the systems, minor variants were also considered (e.g., using lossy filters vs. blocking filters in the coordination patterns). As these variants have very similar size and run times, we display these systems in aggregate and report the variable order generation time, build time and BDD size for the biggest of the variants.

The ResourceManager is built data-abstract, i.e., abstracting from the concrete message values flowing in the system. The Black Box ASK system is an abstracted version of the global ASK model with generic sub-components with non-deterministic behavior, allowing the verification of the top-level coordination patterns.

All computations were performed on a AMD Athlon 64 X2 Dual Core Processor 5000+ (2.6 GHz) with 8GB of RAM, running Ubuntu Linux.

Table 2 shows statistics for the verification of bisimulation equivalence between parts of the ASK model using Vereofy. We checked the correctness of simplifications of the coordination circuitry in the Monks, as well as the equivalence of two different implementations of the ConnectoidWarehouse.

2.5 Lessons Learnt and Conclusion

The subprojects within the validation phase of the *Credo* project have gained the following insights:

System	BDD vars.	Varorder gen. time	Build time	System size BDD nodes	Prop. count	Verification avg.	Verification max.
Thread level							
SchedulerMonk⟨1⟩	241	-	2.3s	18,634	15	0.05s	0.1s
SchedulerMonk_simplified⟨1⟩	215	-	1.8s	10,479	5	0.06s	0.1s
SchedulerMain	30	-	1.3s	158	6	0.01s	0.02
MatcherMonk	135	-	1.4s	2,294	17	0.01s	0.03s
ReceptionMonk	145	-	1.4s	1,863	18	0.01s	0.02s
ResourceManagerMonk⟨1⟩	327	-	3.0s	22,157	15	0.1s	0.2s
UNIXPipelines	29	-	1.3s	193	2	0.01s	0.01s
HappinessValue	26	-	1.3s	33	4	0.01s	0.01s
ConnectoidWarehouse	28	-	1.3s	204	8	0.01s	0.01s
Process level							
Scheduler	494	1,375.0s	15s	68,487	16	2.1s	8.6s
Matcher	274	54.0s	1.3s	9,562	25	0.07s	0.3s
Executer	54	0.5s	1.3s	87	3	0.01s	0.01s
Reception	284	58.0s	4.3s	6,248	19	0.07s	0.4s
Resource Manager	849	3,258.0s	56s	458,199	18	276.4s	4,891.0s
Global level							
Black Box ASK	406	1,225.0s	4.3s	269,360	13	0.9s	1.8s

Table 1: System instances verified using Vereofy, with information about size, properties and run-time information.

System 1	System 2	Bisimulation check time
SchedulerMonk⟨1⟩	SchedulerMonk_simplified⟨1⟩	60s
ResourceManagerMonk⟨1⟩	ResourceManagerMonk_simplified⟨1⟩	319s
ConnectoidWarehouse	ConnectoidWarehouse_verbose	0.1s

Table 2: Bisimulation checking of components in the ASK system.

ReASK. The modeling of ASK with RSL and CARML for the purpose of validation, based on the REO circuits and automata of the final modeling phase, was a straightforward activity, especially due to Vereofy's ability to deal with the model at several levels of abstraction and the compositional and hierarchical nature of the modeling languages. The support for complex structured data domains allows the modeling of complex messages and message dependent coordination between the components.

In Deliverable D6.3, we concluded that the version of the Eclipse REO editor at that time did not adequately support top-down modeling as we did in this subproject. Furthermore, conversion to CARML and RSL for verification with Vereofy needed to be done completely manually. These omissions have been taken into account by the tool developers through the addition of techniques for embedding high-level component specifications in top-level circuits which can be refined in a later stage, and tools to generate CARML/RSL from Reo circuits and the other way around. Sufficient validation of these new possibilities, however, could not be performed.

CreASK. Various techniques have been developed for the simulation, validation and testing of Creol models. We consider all techniques developed thus far (concolic execution, passive trace-based testing) as promising: earlier (published) experiments in the project showed the benefits of applying these techniques, in that they already yielded various flaws in the Creol models developed during the final modeling phase. More methodological support is needed to help the end user in determining which technique should be applied for which purpose. The passive trace-based testing technique will be further evaluated by Almende in the context of its general development activities.

tASK. As we concluded in the assessment of requirement **RC.4**, the UPPAAL and VerifyTA tool are very well suited for the verification of all kinds of timing properties, while the *schedulability analysis technique* covers precisely the aspects necessary for reasoning about the most important non-functionals for ASK. Almende and CWI will probably continue the further automation of property verification with the command line tool VerifyTA – the insights gained thus far in the dimensioning of thread-pools in ASK are already considered valuable, and a lot more is to be expected from the technique and tools. Almende will in the near future concretely apply the insights in new test implementations of the ASK system.

3 Case study 2: Biomedical sensor networks

3.1 Overview of the BSN Case Study

Based on the generic architecture of a biomedical sensor network (BSN) presented in previous deliverables D6.1, D6.1 Addendum, and D6.2, we modelled several aspects and levels of detail using the *Credo* tools: *Creol* (including

several extensions), *UPPAAL*, and *Vereofy*, following the *Credo* Methodology Document [?]. The different final models (I) to (VIII) and their properties are presented in Deliverable D6.3. We chose the AODV routing algorithm as the main subject for our evaluations.[2] In this document, we give an overview of the validation of these models. We compared the results from the evaluation with known results from the real world and other evaluations of the AODV algorithm.

We used the following categories[3] to structure the validation work: (a) *techniques*, (b) *perspectives*, (c) *arrangements*, and (d) *properties*. In this context, *techniques* describe technical measures and procedures to perform the evaluation of a model's properties. *Perspectives* describe the scope of an evaluation, such as (1) observing the behaviour of the entire network configuration, or (2) observing the behaviour of one node. An *arrangement* denotes a set of settings that has an influence of how the model operates; such as using communication failures, timed model, timeouts, energy consumption, and so on.

Functional properties are concrete conditions that can be checked for given arrangements. For our AODV models we have defined a set of properties that we use in the validation process: (A) correct-operation; (B) loop-freeness; (C) single-sensor challenge-response properties; (D) shortest-path; (E) deadlock-freeness (both, for node, and for protocol); (F) and miscellaneous composed system properties. See Annex D6.4.2 [?] for a detailed description of these properties, as well as Annex D6.4.3 and Table 3.

3.2 Validation of the Final Models

We validated the timed automata model (I), the *Creol* AODV model (V), and the flooding and AODV models in Vereofy (VII) and (VIII). Most aspects of the other models from Deliverable D6.3 have been integrated into the *Creol* Model (V), and can be studied there. For a description of Case Study 2, and the final models see Deliverables D6.1, D6.1 Addendum, and D6.3.

3.2.1 Timed automata models of BSN (I).

We modelled message-forwarding in a BSN using timed automata [?], where the sensor nodes communicate using the Chipcon CC2420 transceiver and the IEEE 802.15.4 standard. Based on the model, we have used *UPPAAL* to validate and tune the temporal configuration parameters of a BSN to meet QoS requirements on network connectivity, packet delivery ratio and end-to-end delay. The network studied allows dynamic re-configuration of network topology due to the switching of sensor nodes to power-down mode for energy-saving or their physical movements. Both the *UPPAAL* simulator and model-checker are used to analyse the average-case and worst-case behaviours.

[2] We refer to Deliverable D6.3 and Annex D6.3.3 for further information on AODV and the different models.

[3] The categories (a) to (d) are explained in Annex D6.4.2.

Property	Description	Model		
		(I)	(V)	(VIII)
(A)	Correct Operation	n/a	yes	n/a
(B)	Loop-Freeness	n/a	yes	n/a
(C)	Sgl-sensor challenge-resp.			
(C).(i)	Always send with own ID	n/a	yes	yes
(C).(ii)	msg leads to valid route	n/a	yes	yes
(C).(iii)	RREQ w/o route⇒RREQ bc.	n/a	yes	yes
(C).(iv)	RREQ for me leads to RREP	n/a	yes	yes
(C).(v)	RREP triggers route to originator	n/a	yes	yes
(C).(vi)	RREP is rebroadcasted	n/a	yes	yes
(C).(vii)	send iff route known	n/a	yes	yes
(C).(viii)	routing table integrity	n/a	n/a	yes
(C).(ix)	all msg for sink	n/a	yes	yes
(C).(x)	processing without receive	n/a	yes	yes
(C).(xi)	increasing sequence number	n/a	yes	yes
(C).(xii)	neighbour update triggers	n/a	n/a	yes
(C).(xiii)	updates terminate	n/a	yes	yes
(C).(xiv)	update success	n/a	yes	yes
(C).(xvi)	Rec. in IDLE only	n/a	n/a	yes
(D)	Shortest-Path	n/a	yes	n/a
(E)	Deadlock-Freeness			
(E).(xvii)	node deadlock	n/a	no	yes
(E).(xviii)	protocol deadlock	n/a	yes	yes
(E).(xix)	model deadlock	n/a	yes	yes
(F)	Misc. Composed-System			
(F).(xx)	route stays valid	n/a	yes	yes
(F).(xxi)	only data msg	n/a	poss	yes
(F).(xxii)	NoRERR	n/a	yes	yes
(F).(xxiii)	no useless RREQ	n/a	poss	yes
(F).(xxiv)	RREQ triggers RREP	n/a	poss	yes
(F).(xxv)	# rec. msg.	yes	yes	n/a
(F).(xxvi)	packet loss	yes	yes	n/a
(F).(xxvii)	timing	yes	part	n/a
(F).(xxviii)	network connectivity	yes	yes	n/a
(F).(xxix)	QoS properties	yes	part	n/a

Table 3: Properties evaluated in Case Study 2. Fields marked with "n/a" denote properties that are not applicable; "poss" means that evaluations are possible but were not performed; "part" means that this property only can be partially evaluated.

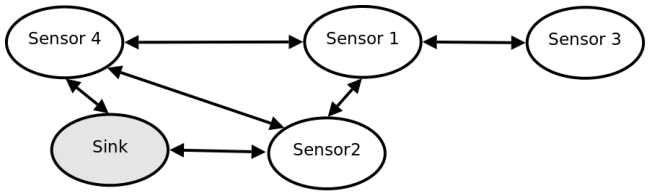

Figure 2: The network used as example in our simulations.

Experiments showed that this model can be used for efficient simulation and parameter tuning. The properties used to validate this model include dynamic network topologies (F).(xxviii), tuning and verification of QoS properties (F).(xxix), the absence of deadlocks (E).(xviii), network connectivity (F).(xxviii), and packet delivery ratio (F).(xxv) and (F).(xxvi). Note that Model (I) does not implement flooding nor AODV, and cannot be compared with the other models with respect to functionality. The validation details of Model (I) are shown in Annex D6.2.3 of Deliverable D6.2 [**?**].

3.2.2 *Creol* Model of AODV (V).

Model (V) [**?**] of the AODV algorithm for BSN needed an extension to the *Creol* language, which lead to the definition of *CreolE* (Extended Creol) [**?**] shown in Annex D6.4.1. The updated model (V) now integrates both flooding and AODV into one model, including aspects of Models (II), (III), (IV), and (VI). Below, we briefly describe the validation of Model (V); a more detailed version can be found in Annex D6.4.2.

We introduced the dimensions of *techniques*, *perspectives*, *configurations*, and *properties* for this evaluation. The functional properties used for this evaluation were divided into five property classes (A) to (F). All these properties are aligned with the properties used to evaluate the *Vereofy* tool [**?**] for a later comparison. We performed network simulations of the composed system, and component testing of a single node in order to evaluate these functional properties. Different properties are suited for simulation and component testing.

For our evaluation of the properties we simulated using techniques such as auxiliary variables, and assertions. Most of our experiments used a network via symmetrical communication with four sensor nodes and one sink node, as shown in Figure 2. We simulated the AODV model using various configurations in order to validate the model for different situations. We looked at reliable networks, lossy networks, timeouts, energy consumption, and timed modelling. We checked selected properties from classes (A), (B), (D), (E), and (F) that are suited for simulating the composed network. We present some of the evaluations below.

Reliable communication: As long as the network is connected[4], the evalu-

[4]We also simulated networks that are not connected, which behaved as expected.

ations showed that the modelled AODV algorithm fulfils the properties (A), (B), (D), (E), and (F). We emphasised on the evaluation of packet loss (F).(xxvi), and loop-freeness assertion (B). Other predicates for loop-freeness were also used[5], and small, faulty changes in the model were introduced (which led to expected failures of Property (B)). The shortest path property (D) was fulfilled in all simulated occasions.

Lossy communication: When simulating lossy communication, both for singlecast and for broadcast messages the packet loss ratio (F).(xxvi) increased as expected. We also could observe an increased number of RREQ and RREP messages in the system, using auxiliary variables. The shortest path property (D) was fulfilled in all simulated occasions.

In one occasion we could observe that the loop-freeness property (B) was not fulfilled using lossy communication. However, we have not yet investigated the reason of this failure. We have saved the configuration for further investigation, and the state stored in the *Maude* file.

Re-sending lost messages with timeouts: The model allows to configure re-sending of lost RREQ messages up to a certain number of times, using a timeout mechanism. We could observe that this mechanism decreased the packet loss ratio (F).(xxvi), but at the same time we can state that this mechanism does not avoid all packet loss.

Energy consumption: Using the energy consumption configuration we can force a communication failure of certain nodes after some actions. Using this configuration we can study the re-routing behaviour in detail. We also studied the packet loss ratio (F).(xxvi) for configurations where nodes fail due to energy consumption.

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 ratio (F).(xxvi) is different from the untimed case. This behaviour is expected.

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

We did not evaluate the properties (F).(xxi), (F).(xxiii), and (F).(xxiv), since it is necessary to store all messages during the simulation. However, such a configuration will lead to a high number of states (state explosion).

To evaluate the single-sensor properties (C), we employed component testing, where the network is replaced by a test harness, and only one node under test is

[5]These predicates were designed to fail in order to have examples of predicates that are supposed to fail. The *Creol* tools behaved as expected.

used. This evaluation is performed by studying the output messages of a node when given input messages are applied. Communication between nodes happens always through the interfaces of the network object that in turn communicates with other nodes. Through these interfaces, messages are sent to the node under test, and the reactions from this node are investigated.

A test verdict is reached by running the test harness in parallel with the object under test. 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. If the test harness *deadlocks*, it expects a message from the object under test that is not arriving 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.

In addition to domain-specific single-object properties that have to be tested, test cases can be generated using Model (VIII) and the *Vereofy* tool. In this case traces received from the node under test are tested against *message patterns*, i.e. we abstract away from details that could lead to spurious test failures not expressing a malfunctioning system. The 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. No tool support was implemented for this test technique yet, but since *Vereofy* traces contain all the needed information to simulate an environment for the node under test, implementation is considered to be straightforward.

3.2.3 Flooding (VII) and AODV in Vereofy (VIII).

For the Case Study 2 we were able to successfully model two variants of communication protocols used in the context of the biomedical sensor networks. In the early project stage, a variant of a flooding protocol covering possible dynamics in the network topology was specified with the help of CARML and RSL. For this model we were able to compose and verify network structures having more than 10 sensor nodes.

In the second stage, the AODV protocol became the object of our investigations for all partners. Modelling this required complex messages to cover all the necessary information that has to be transmitted between the individual sensor nodes, as well as highly complex CARML specifications of the sensor nodes themselves, with routing table storage, complex rules for the correct maintenance of the routing table information and necessary message handling and generation. A detailed description of the Vereofy models for flooding and AODV can be found in D.6.3.

Although CARML and RSL were designed to specify coordination aspects and inter-process communication, it was still possible to model the needed computation features with the help of Vereofy's input languages. Moreover, we were able to find errors in the model, fix them and show functional correctness properties of the protocol implementation for a single sensor node and for simple network structures. For the later is was helpful to apply abstraction and simplification methods to create simpler variants of the model (D5.6), with less data

dependencies, where the significant properties of the routing protocol are still holding. E.g., for static network topologies the freshness of routing information has no bearing on the decision if and when routing tables should be updated. Using iterative building techniques[6] (D5.6) we were able to create random traces for larger network structures to be explored by the user.

While working with the model, we encountered misbehaviour and bugs in the model. We list some of them below. A detailed list of properties, including the ones mentioned above, can be found in Annex D6.4.3 (Properties for Validation of Case Study 2), as well as in Table 3.

1. RREP messages were not correctly forwarded: we figured out that RREP messages were not forwarded in the right direction. Instead of forwarding them they were sent back to the originator of the reply. Thus, no routes longer than a single hop were computed correctly.

2. Repeating RREQ messages may flood the network and lead to a protocol deadlock: The original protocol contains a timeout for RREQ messages. They are re-sent when they receive no answer in time. We had to remove this in our time-abstract model to avoid flooding the network with RREQ messages and blocking all other communication. We found this problem while looking for possible deadlock situations.

3. Whenever a link failure to a neighbour occurred the route for this node was marked invalid, but other routes, which use this dead node as a next hop, were not marked as invalid. Thus, messages could be addressed to dead nodes, causing new link failures. The new update procedure works recursively on the routing table and ensures this property.

For checking properties of network structures with a few sensor nodes we pre-computed variable orderings to benefit from preferably compact BDD representation of the system behavior. The pre-computation itself consumed a few hours and the system composition afterwards was then possible to be done within minutes. With an increasing number of sensor nodes, more BDD-variables are needed for encoding the addresses inside of messages, as well as message buffers, and the routing table of each node. Table 4 illustrates how many BDD-variables would be needed for the encoding of a) a single data item, b) the states of a single sensor node, and c) the states of the composite system. At the end of the project, we now are able to compose and verify AODV network structures with up to three nodes with a state space of about 10^{23} reachable states, which is quite a success for a new tool. Handling larger numbers of sensor nodes is still possible, but will require further development and optimisation, e.g., of abstraction techniques and the BDD-representations, in the future.

[6] The transition function is not build for all possible configurations of the system, but only for a subset or a single state. Consecutively applied, first for an initial state and then for randomly chosen successors, leads to a random trace of the system.

boolean variables	AODV_2	AODV_3	AODV_4	AODV_5
data values	15	18	20	23
sensor node states	58	74	91	111
system states	116	222	364	555

Table 4: Number of boolean variables needed for the encoding in the Vereofy model of AODV

Required properties	**Need to have**	**Nice to have**	**Dropped**
(α) Timing	Max/Min end-to-end delay	Average end-to-end delay	Channel access delay, propagation delay
(β) Network throughput		Max/Min/Average throughput	
(γ) Packet delivery ratio	Requirements on packet delivery ratio	Average end-to-end ratio	
(δ) Network connectivity	Network deadlock, isolated node	Network Bottleneck	
(ε) Energy consumption		Node and network lifetime	
(ζ) Memory and Buffer	Buffer overflow possibility	Memory consumption	
(η) Wireless channel	Collision possibility	Channel access failure, average bit error rate	Channel efficiency/deviation
(θ) Mobility	Validation of routing protocols and local topology changes	Impact on delay, throughput, and packet delivery ratio	
(ι) Interference	Concurrent transmitting		Environmental interference, thermal noise

Table 5: Required properties and priorities

3.3 Assessment of Initial Requirements

Deliverable D6.1 Addendum lists eight elements to define the scenario. Over the course of the project the scenario has become more abstract. In addition, the focus for our work has moved from the overall case to investigate the AODV routing algorithm.

The table of required properties and their priorities has been given in Deliverable D6.1 Addendum. We repeat the table of properties in Table 5, and the requirements (α) to (ι) are listed and marked with priorities.

Max t steps	Energy	Msg loss behaviour	Timeout behaviour	#Creol lines	#Maude lines	# rewrites	time (ms)
500	x	none	never	1579	1026	9 444 821	17 123
5000	x	none	never	1579	1026	62 841 821	114 808
500	x	every 10th	never	1593	1035	10 709 994	19 552
500	x	every 10th	every 10th	1601	1039	12 112 216	22 311
untimed	50	every 10th	every 10th	1587	1011	11 647 148	17 969
500	50	every 10th	every 10th	1636	1058	8 305 408	15 548

Table 6: Table showing code size and run-time for the tested cases with 5 nodes.

The Credo-tools are suited to evaluate the existence and non-existence of certain properties. To a lesser extent they can count events, and find minimum/maximum values of properties. In simulation mode, e.g., using a Monte-Carlo-type setup with many simulations, averages can be computed.

In Model (I), which implements forwarding of messages on a lower network layer, certain aspects of the following required properties are implemented: timing (α), network throughput (β) (to a certain extent), packet delivery ratio (γ), network connectivity (δ), collision in wireless channels (η), and interference for concurrent transmissions (ι).

In Model (V), the following required properties are implemented: timing (α) (to a minor extent), packet delivery ratio (γ), network connectivity (δ), aspects of energy consumption (ε), aspects of memory and buffer consumption (ζ), and the effect of topology changes (θ). Some aspects of collision possibilities (η) and concurrent transmissions (ι) can also be evaluated, but to a lesser extent than originally desired.

In Model (VIII), the purely functional properties for network connectivity (δ), buffer overflow possibility (ζ), and topology changes (θ) have been verified.

3.4 Quantitative Evaluation

In the following we outline the sizes and dimensions used in the Case Study 2, and the aspects and properties checked, as well as indications to runtime and type of machines.

For Model (V) we show code size, run-time, and rewrites for an AODV model of five nodes in Table 6. We varied the number of time-steps, the energy consumption, and the message loss behaviour. All diversifications are contained in one large model of about 1700 lines of *CreolE*. Note that the size difference between these diversifications is rather neglectable. For each of the cases we measured the size of the resulting compiled artefacts in *Creol* and Maude, respectively. We also recorded the number of rewrites and the execution time on a desktop PC with an AMD Athlon 64 Dual core processor with 1.8 GHz.

We also experimented with varying the number of nodes. We measured 32484226 rewrites (14868ms)[7] for five nodes, and 90468655 rewrites (40978ms)

[7] These measurements were performed on a computer with an Intel Core2 Duo CPU E8400 with 3 GHz.

System	BDD vars.	Varorder gen. time	Build time	System size BDD nodes	Prop. count	Verification avg.	Verification max.
Sensor Node(1,k)							
$k = 2$	106	10.2s	0.9s	5,739	27	10.4s	67s
$k = 3$	131	44.0s	1.4s	26,598	17	87.6s	163.0s
$k = 3$, no Fix2	134	51.0s	1.4s	21,513	2	521.5s	536.0s
Network (k Nodes)							
$k = 2$	212	253.0s	5.6s	233,619	21	1.94s	7.3s
$k = 2$, buffered	240	2,001.0s	9.8s	629,559	1	2.1s	2.1s
$k = 2$, buf., no Fix1	238	1,481.0s	10.1s	816,450	1	23.0s	23.0s
$k = 3$, abstr.	242	15,300.0s	4.7s	2,129,780	3	909.0s	2,304.0s

Table 7: System instances verified using Vereofy, with information about size, properties and run-time information.

for six nodes for an untimed model, about 10% packet loss without timeout behaviour. However, when varying the settings, e.g., we introduce the timeout behaviour, we could observe that in some cases the run-time for six nodes is much less than for five nodes. While this might sound strange, this can be explained that rather small changes in topology can have a substantial impact on the behaviour of the algorithm.

Table 7 shows statistics for the RSL/CARML model of the AODV protocol, Model (VIII), as used by the Vereofy model checker. The first part deals with verification of the individual sensor nodes (with identifier 1) in the context of a network of k sensor nodes, i.e. with a routing table for k nodes. The second part deals with networks of k connected sensor nodes. In addition to the standard models, some variants are considered for certain properties. "Buffered" denotes networks where there is additional buffering of the messages flowing through the network. The two systems with "no Fix" refer to variants of the model where certain bugs are still present which were fixed using the verification results. In the case of the 3 nodes network, some abstractions are applied, i.e. to the handling of sequence numbers and link failures.

The first column of the table shows the number of boolean BDD variables used to encode the system. The next column shows the time spent to calculate a suitable variable ordering for the system. After such a variable ordering is calculated, it can be reused, with the next column showing the time spent to build a symbolic representation of the system using the previously calculated variable ordering. The next column shows the number of BDD nodes used for the symbolic representation of the transition function (generally the most complex part) of the system.

The next three columns then show the number of temporal logic properties (see Technical Annex D6.4.3 for details) checked for the given systems and the average and maximum runtime of the model checker used to verify or falsify the properties.

All computations were performed on a AMD Athlon 64 X2 Dual Core Processor 5000+ (2.6 GHz) with 8GB of RAM, running Ubuntu Linux.

3.5 Lessons Learnt and Conclusion

The *Credo* tools based on *Creol* (*CreolE*), *Vereofy*, and *UPPAAL* offer different ways of modelling, supporting different techniques, perspectives, arrangements and evaluation of properties. After validating forwarding in a wireless sensor network modelled in Model (I) in regard to QoS properties, loss rate, and parameter tuning, we decided to evaluate the functional aspects of routing algorithms; we selected the well-known AODV algorithm for this task.

Using the network simulation of Model (V), several arrangements were evaluated, where most of the properties hold as expected. In some occasions, we found properties that did not hold in the simulation, either due to bugs in the model, properties of the modelled AODV algorithm, artificially introduced bugs in the code, 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. This problem could be recognised and fixed later on.

Using component testing of the *Creol* model of a single sensor node, we validated the model's behaviour with respect to properties important for the correct functioning of the AODV algorithm. We created a test driver simulating a network environment and a method of abstracting away parts of messages for the purpose of reaching a test verdict. No incorrect behaviour of the node model was identified during single-object testing, which was not surprising since the model had already been extensively exercised during model creation and initial simulation experiments. The resulting test suite served well for regression testing during further modification and enhancements of the model.

We modelled a highly distributed application with many autonomously acting objects (sensor nodes). While evaluating the properties of the AODV algorithm, we encountered several challenges, including modelling a suitable abstraction, using suitable language constructs of *Creol*, and observing the properties from a suitable perspective. The major challenge when evaluating the AODV algorithm form a network perspective is to avoid a high number of states (state explosion) in the underlying interpreter. We see that the properties suitable for component testing are disjunct from the properties suitable for network simulation. Therefore, these techniques are complementary to each other.

Using the inter-process communication features of CARML and RSL we were able to model both flooding and AODV. Using *Vereofy* we were able to detect errors in the model, fix them, and show the functional correctness properties of the protocol implementation for a single sensor node and simple network structures. Using iterative building techniques, we were able to create random traces for larger network structures, and provide some traces for the component testing of Model (V).

We found the *Credo* languages and tools useful in the evaluation of the AODV algorithm, and in order to get insight into how complex algorithms like AODV work. We observed that small changes in the algorithm, and in chosen arrangements imply changes in its behaviour. We also detected the breach of certain properties, that will lead to further investigation of this misbehaviour, its removal and, eventually, to a better understanding of AODV and other

algorithms used for sensor networks.

Technical Annexes

This document contains the following technical annexes:

D6.4.1: This technical annex[?] gives an overview of the *CreolE* language, an extended version of the *Creol* language based on best practices gained during the *Credo* project.

D6.4.2: In this technical annex[?], the validation of a Creol model of AODV is performed by evaluating functional properties using simulation and component testing. The annex also explains the categories (techniques, perspectives, arrangements and properties) which have been used to structure the validation phase of Case Study 2 (BSN).

D6.4.3: This technical annex lists all the properties used in the validation of Case Study 2 (BSN) with *Vereofy*.

D6.4.4: In this technical annex, a detailed description of the models used for the schedulability analysis performed in case study 1 (ASK) is given.

D6.4.5: This technical annex lists all the properties used in the validation of Case Study 1 (ASK) with *Vereofy*.

D6.4.6: In this technical annex, an update is given of the final models (Reo networks and automata) of the ASK system, based upon the results of the validation phase.