Testing the Cooperation of Autonomous Robotic Agents

Raimar Lill and Francesca Saglietti
Informatik 11 - Software Engineering, Friedrich-Alexander-Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, Germany

Keywords: Autonomous Systems, Coloured Petri Nets, Genetic Algorithms, Optimized Test Case Generation, Structural Testing.

Abstract: This article proposes an approach to testing the cooperative behaviour of autonomous software-based agents with safety-relevant tasks. It includes the definition of different model-based testing criteria based on the coverage of Coloured Petri Net entities as well as the automatic generation of appropriate test cases. The multi-objective optimization problem considered addresses both the maximization of interaction coverage and the minimization of the amount of test cases required. The approach developed for its solution makes use of genetic algorithms. The resulting automatic test case generation process is presented in this article together with the experiences gained by applying it to cooperating autonomous forklifts.

1 INTRODUCTION

1.1 Motivation

Modern software-based applications increasingly rely on de-centralized functionalities distributed among entities. While classical component-based software usually involves behavioural synchronicity, however, autonomously cooperating agents are conceived to take individual decisions on the basis of their local sensorial perception and reasoning capabilities (Saglietti, Söhnlein and Lill, 2011). Typical application domains addressed by autonomous cooperation include mobile robots or traffic control based on car-to-car communication. Evidently, local decisional autonomy and shared cooperative tasks allow for higher flexibility and performance than central controllers; on the other hand, the resulting global behaviour induced by autonomous decisions involves also a much higher variety of potential interaction scenarios. In particular, this multiplicity poses serious challenges to verification, as compositional testing or proving techniques relying on separation of concerns cannot be taken to provide adequate evidence any longer.

In fact, the potential failure behaviour of autonomous cooperative robots goes beyond the possibility of incorrect performance of one entity including also inappropriate decision-making due to inaccurate perception instruments, inadequate interpretation of signals perceived, incorrect identification of actions required or inconsistence between decisions of cooperating agents.

Therefore, in order to improve the state-of-the-art, a model-based approach for testing cooperating autonomous systems was developed within the European ARTEMIS project R3-COP. It aims at capturing the inherent interaction multiplicity by an appropriate modelling notation, from which to derive representative test cases. In more detail, the approach developed is based on the following steps:

- modelling of the behaviour of a cooperating autonomous system;
- definition of adequate coverage criteria based on the modelling elements of the notation chosen;
- automatic generation of model-based test cases achieving given coverage criteria.

The test data generation process follows a multi-objective optimization strategy based on genetic algorithms (Mitchell, 1996): in fact, it aims at maximizing test coverage while minimizing the number of tests involved.

After addressing related work in the next section, the rest of the article is structured as follows:

- *chapter 2* briefly outlines the benefits of using Coloured Petri Nets (CPNs) for the purpose of modelling cooperating autonomous systems; furthermore, objectively reproducible coverage criteria based on CPN modelling elements are presented and hierarchically organized;

Lill R. and Saglietti F..
Testing the Cooperation of Autonomous Robotic Agents.
DOI: 10.5220/0004990402870296
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 287-296
ISBN: 978-989-758-036-9

- *chapter 3* proposes an incremental testing procedure consisting of successive testing phases characterized by gradually increasing levels of contextual detail;
- *chapter 4* is devoted to the automatic generation of optimized CPN-based tests by means of 2 different optimization strategies: the former addresses conflicting objectives by pre-defined target priorities (section 4.1), while the latter one allows to capture varying target priorities by moving along a whole Pareto front (section 4.2);
- finally, *chapter 5* illustrates the practicability of the technique developed in the light of an example inspired by a real-world application involving cooperating forklifts.

1.2 Related Work

The approach presented in this article is based on the classical concept of model-based testing (e.g. Utting and Legeard, 2007; Broy et al., 2005) allowing for the extraction of significant test cases from dedicated behavioural and environmental models. In order to capture the behaviour of cooperating autonomous systems, it makes use of the modelling language CPNs (Jensen and Kristensen, 2009). The coverage criteria proposed in section 2.2 were partly inspired by already existing coverage concepts (Zhu and He, 2002) for Predicate-Transition Petri Nets (Genrich and Lautenbach, 1981). For the purpose of automatic test case generation, they were transferred to CPNs giving rise to appropriate metrics for the evaluation of the fitness of candidate test case sets.

Alternative approaches (Nguyen et al., 2012) and (Micskei et al., 2012) were also devoted to the automatic generation of test cases for autonomous software agents by means of evolutionary techniques. They differ, however, from the target pursued in the present article by focusing on testing for robustness in terms of aiming at the generation of exceptional test scenarios, e.g. involving unusually high stress, human misuse, communication anomalies, behavioural extremes etc. Such approaches assume the previous identification of anomalous behaviour; this may be hard to be achieved in general, especially in case of numerous interacting agents. In addition, they do neither address the global amount of behavioural multiplicity captured by interacting autonomous systems, nor the amount of testing required; both these objectives, on the other hand, are pursued by the technique developed in this article.

2 MODEL-BASED TESTING USING CPNS

2.1 CPNs for Modelling Autonomous Cooperating Agents

In order to capture the high behavioural multiplicity of interaction scenarios arising from autonomous cooperation, an adequate modelling notation is required. *Coloured Petri Nets* (Jensen and Kristensen, 2009) have proven to be particularly useful for this purpose thanks to their capability of providing a compact and scalable representation (Lill and Saglietti, 2012a).

Classical Petri Nets are well-known techniques for modelling and analyzing concurrent processes, in particular capturing their cooperative behaviour as well as potential conflicts. CPNs result by enriching the tokens of ordinary Place/Transition Petri Nets (Murata, 1989) with type-specific data values (*colours*). For this purpose, each CPN place is assigned a *colour set* specifying the type of tokens that may be allocated to that place. At any time, the net marking defines the current *state*.

In order to control the production and consumption of coloured tokens, CPN arcs are assigned dedicated *arc expressions* determining a multi-set of coloured tokens to be produced resp. consumed during transition firings. In order for a transition to fire, each variable occurring in its input arc expressions has to be bound to a specific colour such that a sufficient number of tokens of that colour (determined by evaluating each input arc expression) is available in the corresponding input place. The firing of a transition w.r.t. an enabling *variable binding* is denoted as an *event*. Each event leads first to the consumption of tokens in each input place of the transition in amounts and colours as indicated by evaluating the corresponding input arc expression. After the firing, tokens are produced in output places in amounts and colours as indicated by evaluating corresponding output arc expressions of the transition. *Transition guards* may further restrict the firing of a transition by requiring the fulfilment of given conditions.

The sound mathematical basis on which CPNs are based allows for the use of formal analysis techniques (Jensen and Kristensen, 2009). Furthermore, dedicated tools support step-wise simulation and state space analysis (Jensen, Kristensen and Wells, 2007; Westergaard and Kristensen, 2009). While sharing with classical Petri Nets the benefit of providing an intuitively appealing visualization of complex processes, CPNs offer

additional advantages by supporting full expressive power thanks to the concept of coloured tokens and net annotations in SML (Milner et al., 1997). By permitting to encapsulate data information within the type-specific tokens, CPNs can be easily adapted to meet application-specific requirements without needing to change the underlying net layout, hereby supporting compactness and scalability. Further extensions of CPNs also support hierarchical design and timely aspects (Jensen and Kristensen, 2009).

2.2 Hierarchy of CPN-based Coverage Criteria

In the following, a CPN-based *test case* is defined to be a pair consisting of an initial CPN state and of a finite sequence of CPN events. For the purpose of providing objectively reproducible test stopping rules based on measurable test targets, the following cooperation-tailored coverage criteria were introduced in (Lill and Saglietti, 2012b) on the basis of CPN *model entities* (i.e. *transitions*, *events* and *states)* to be covered during testing.

Transition-based Coverage Criteria address the verification of generic, though non-trivial system functionality of robots (e.g. basic motor activities or self-localization). By limiting the testing scenarios to the mere transition level, the multiplicity of data flow is deliberately kept out of the testing scope by focusing on one single action instance. Testing criteria addressing this relatively coarse level of abstraction include the "*all transitions*"- criterion demanding the activation of each single transition, the "*all transition pairs*"- criterion demanding in addition also the triggering of all possible pairs of transitions, as well as the "*all transition sequences*"- criterion extending the previous criteria to include the firing of any possible sequence of transitions.

Event-based Coverage Criteria go beyond single generic transitions by explicitly addressing the whole variety of possible action instances (e.g. varying robot movement scenarios under different terrain conditions). By focusing on event occurrences, the underlying data flow multiplicity - including the amount and the colours of tokens - is intentionally integrated into the testing scope. Coverage criteria addressing this finer level of abstraction include the "*all events*"- criterion demanding the occurrence of every event, the "*all event pairs*"- criterion demanding in addition also the occurrence of any possible pair of events, as well as the "*all event sequences*"- criterion extending the previous criteria to include the occurrence of any possible sequence of events.

State-based coverage criteria, on the other hand, enrich the accuracy of test observations by distinguishing between different operational conditions present before and after event occurrences, such as potential mid- or long-term interferences with other robots or obstacles. In terms of model entities, this is captured by addressing different token amounts and colours between and after action instances. Coverage criteria at this particularly fine level of detail are the "*all states*"- criterion demanding the traversal of every state, the "*all state pairs*"- criterion demanding in addition the traversal of any possible pair of successive states, as well as the "*all state sequences*"- criterion extending the previous criteria to include the traversal of any possible sequence of states.

As event-based and state-based coverage criteria require the construction of the underlying CPN state space graph, the test case generation may be limited by state space explosion (Valmari, 1998). This problem, however, may be partly circumvented by the following strategies (Pelánek, 2009):

- using parallel or distributed computing for calculating the state space graph;
- reducing the state space, e.g. by addressing a higher degree of abstraction or by restricting the model parameters to tractable dimensions considered as acceptable for testing purposes;
- restricting the coverage targets to operationally relevant portions of the state space.

Under the realistic assumption that all CPN transitions are connected by arcs, all guards are satisfiable and every state can be reached from another state via at most one event, under a given initial marking the causal implications between the coverage criteria introduced above are illustrated by the subsumption hierarchy shown in Figure 1 (Lill and Saglietti, 2012b).

3 INCREMENTAL STRUCTURAL TESTING

In analogy to conventional structural software testing, the scope of the testing object may be step-wise enriched by growing amounts of contextual details. In fact, the hierarchy shown in Figure 1 highlights the increasing refinement from generic actions (bottom level) via specific action instances

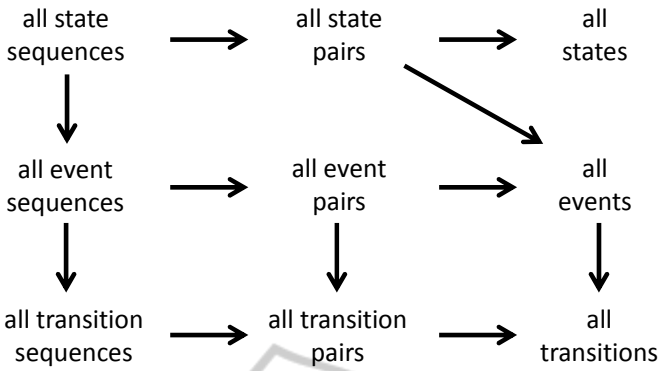

Figure 1: Subsumption hierarchy of CPN-based coverage criteria.

(intermediate level) to action instances involving different operational pre- and post-conditions (top level). Based on this concept the following structural testing phases for cooperating autonomous systems are proposed:

1. **local** test of **context-free** actions represented by CPN *transitions*, intended to verify generic, but non-trivial robot functionalities like basic motor activities by robots or self-localization functionalities by sensors.
2. **local** test of **context-specific** actions represented by CPN *events*; after that, the second testing phase extends the testing scope to the coverage of specific action instances. For example, motor activities should be tested on different terrain conditions (e.g. varying grip or slope) and self-localization should take place in different environments (e.g. outdoor or in a closed factory room).
3. **global** test of **contextual** system behavior represented by CPN *state pairs*. Therefore, a third phase also takes into account global system states encountered before and after event occurrences. For example, the actions of mobile robots could be affected by weather conditions, battery status or the interference of other robots or obstacles. Post-conditions encountered after event occurrences may often lead to further conflicts (e.g. robots blocking each other). This may be exploited for the prolongation of test cases in order to capture these situations.

While these structural testing phases can systematically support preliminary verification activities aimed at fault detection, they evidently do not allow for the quantitative assessment of operational reliability, as they exclusively address structural coverage ignoring issues like frequency of occurrence or criticality of events. Therefore, if required, these phases should be completed by reliability testing based on an operationally representative usage profile.

4 OPTIMIZED TEST CASE GENERATION

Automatic model-based test case generation pursues two main objectives:

- a fault detection benefit achieved by maximizing interaction coverage;
- an economic benefit achieved by minimizing the amount of test cases.

In general, these objectives are conflicting, as higher coverage usually demands for more test cases. The underlying multi-objective optimization problem is approached by evolutionary techniques (Freitas, 2002). Inspired by Darwinian evolution theory, they rely on the successive improvement of solution candidates by genetic operators (s. Figure 2).

In the context of CPN-based testing, each *individual* within a given *population* is intended to represent a candidate set of test cases, where test cases are referred to as *genes*.

After a starting population is randomly generated, each of its individuals is evaluated in terms of its achievement of the objectives by a so-called *fitness function*. The following operators are then applied for the purpose of generating a subsequent population of given size.

The *elitism* operator transfers a fixed percentage of the population consisting of the best-fitted individuals unaltered to the successive population.

Furthermore, all individuals are considered for *recombination* where the recombination process is

based on a *selection* operator repeatedly choosing pairs of individuals according to given strategies and on a *crossover* operator recombining them to build two new individuals. In order to reduce the risk of worsening very fit individuals by recombination, a fixed percentage of the elite skips this phase and directly proceeds to the successive mutation operator. Two crossover operators were defined and implemented:

- **uniform crossover:** exchange of test cases between selected individuals; for each test case of an individual, the probability of being transferred to the other is decreasing with coverage progress in order to avoid weakening very good individuals towards the end of the optimization procedure;
- **cut & glue:** recombination of two single test cases from selected individuals by randomly splitting each of them into two parts in such a way that processing both initial parts results in a common state; the sequential endings of both test cases are then interchanged resulting in two different and meaningful test cases.

In order to increase genetic diversity, test cases building the new population may be successively subject to *mutation* operators, at an operator-specific probability depending on the coverage progress currently achieved. The intention is to take into account the varying need for further genetic material throughout the generation process, tending to increase this material at the beginning while reducing it towards the end. In more detail, the mutation operators applied are the following:

- the **add** operator (abbr. by a) inserts a randomly generated new test case;
- the **delete** operator (abbr. by d) removes a randomly chosen test case;
- the **replace** operator (abbr. by r) concatenates the delete and add operator;
- the **modification** operator (abbr. by m) alters a test case by replacing its events (starting from randomly chosen intermediate state) with potential randomly chosen alternative events up to a pre-defined test case length resp. up to the reaching of a final state.

In order to prevent the loss of precious genetic material during evolution, for each entity a set of test cases covering it is stored in a so-called *gene pool* from which the add operator may successively access for re-insertion.

The algorithm continues as long as no individual candidate fulfils given stopping criteria, e.g. in case a minimum fitness value is achieved or the number of iterations exceeds a certain limit.

4.1 Weight-based Optimization

For calculating the fitness of each individual, a fitness function has to be specified. One test case generation technique involves the definition of dedicated weights for each mission objective.

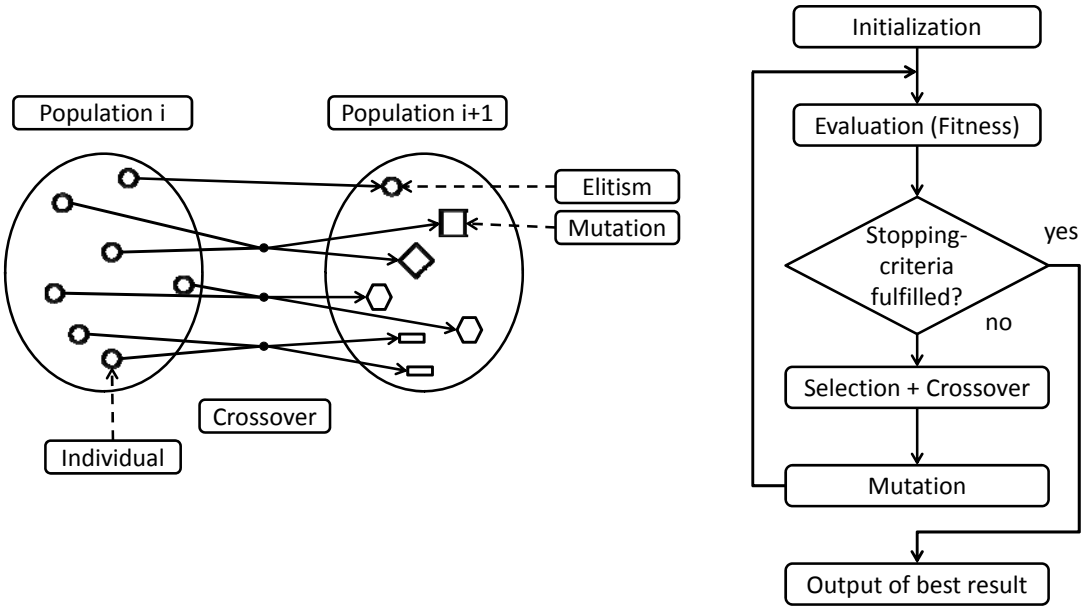

Figure 2: Genetic operators (left) and scheme of a genetic algorithm (right).

The fitness of each individual ts is based on two measures capturing the achievement of both objectives.

The degree of fulfilment of the first objective (coverage maximization) is evaluated by the relative coverage c(ts) achieved by a test case set ts:

$$c(ts) = \frac{\text{number of entities covered by ts}}{\text{total number of entities}}$$

On the other hand, the degree of fulfilment of the second objective (test number minimization) is evaluated by the following normalized value s(ts):

$$s(ts) = \frac{size_{max} - size(ts)}{size_{max} - size_{min}}$$

where

- $size_{max}$ is the size of the largest test case set in the current population;
- $size_{min}$ is the size of the smallest test case set in the current population;
- size(ts) is the size of the test case set under evaluation.

Evidently, the higher the value s(ts), the lower the size of the corresponding test case set.

Depending on the relative priority of each objective, the overall fitness of test case set ts is evaluated by the following weighted sum:

$$fitness(ts) = w_1 \cdot c(ts) + w_2 \cdot s(ts)$$

where w_1 and w_2 are weights with $w_1 + w_2 = 1$.

Before recombining individuals, a so-called elitism operator transfers a fixed percentage of individuals with best fitness values unaltered to the following population ensuring that populations do not degrade over time.

4.2 Pareto Optimization

Using weighted sums to determine fitness values has the disadvantage that the tester is obliged to define the objective weights before the optimization procedure. An alternative solution not requiring any a priori weights is offered by Pareto optimization where a solution is called Pareto-optimal if no other solution has a better rating with respect to all objectives. A set of Pareto-optimal solutions is referred to as a *Pareto front*.

In case of Pareto optimization, the generation process no longer aims at achieving one best-fitted solution, but rather a Pareto front offering optimal solution alternatives. The stopping rules of genetic evolution must be adapted accordingly: the algorithm terminates after reaching a maximum number of iterations or as soon as the Pareto front is stable, i.e. it is maintained unaltered for a pre-defined number of iterations.

For assigning fitness values, the algorithm proceeds by repeatedly extracting sets of non-dominated individuals from the current population. This results in a ranked sequence of Pareto fronts; all individuals of the same front are then assigned the same fitness value which decreases with the rank of the front.

The elitism operator has to be adapted such that the best Pareto front (i.e. the first-ranked set of non-dominated individuals) is transferred unaltered to the following population. In order to allow for a genetic evolution, if the elite front consists of more than half of the original population, the size of the next population is increased such as to contain twice as many individuals as the elite.

5 PRACTICAL APPLICATION

The model-based test generation process proposed was applied to a CPN modelling the cooperation of autonomous forklifts moving within a logistic warehouse (s. Figure 3). In more detail, an arbitrary number of robotic agents move along a narrow lane consisting of an arbitrary number of discrete segments.

The robots are assigned missions (transition *next order*) by a central controller and aim at accomplishing them as autonomously as possible.

If possible, robots proceed towards their designated target segments by moving along the lane (transitions *forward* resp. *backward*, depending on their direction). In case they recognize passive obstacles, they stop in order to avoid collisions.

Robots moving in different directions and meeting each other cooperate by switching positions (transition *switching maneuver*). If unable to access a specific segment (e.g. due to a slow preceding robot or a passive obstacle), robots raise an alarm (transition *traffic holdup*) after 5 unsuccessful attempts to access the segment. Successfully completed missions are logged in the order of their accomplishment (transition *mission completed*).

The two optimization strategies addressed in this article have been implemented in the Java framework Access/CPN (Westergaard and Kristensen, 2009).

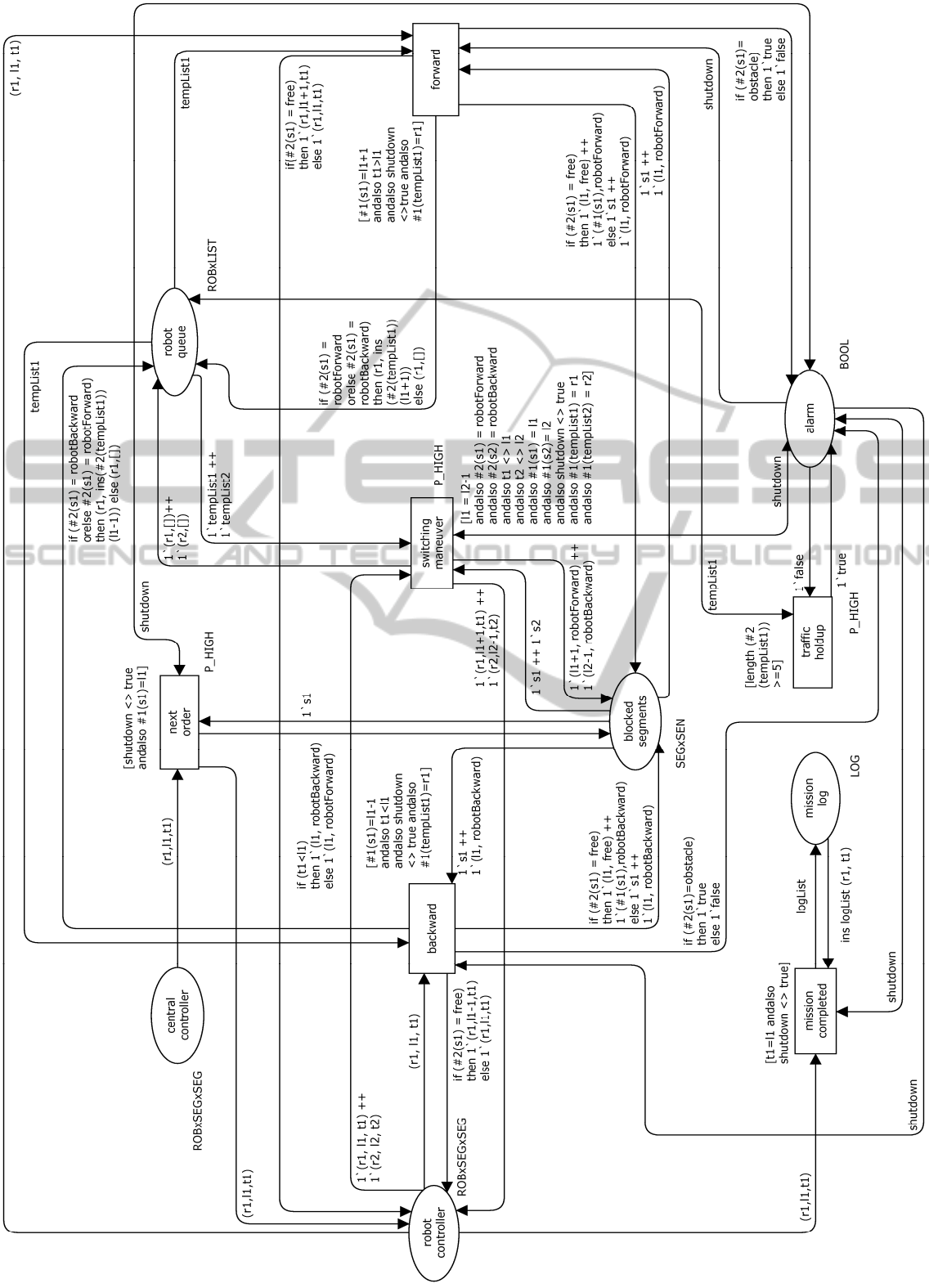

Figure 3: CPN model of cooperating forklifts.

Table 1: Parameterization of the genetic algorithm.

<table>
<tr><td colspan="2" rowspan="2">initialization</td><td colspan="2">population</td><td colspan="2">individual</td></tr>
<tr><td colspan="2">10 individuals</td><td colspan="2">1 test case</td></tr>
<tr><td colspan="2" rowspan="2">stopping criteria</td><td colspan="2">weight-based optimization</td><td colspan="2">Pareto optimization</td></tr>
<tr><td>≤ 1000 iterations</td><td>fitness = 1.0</td><td>≤ 1000 iterations</td><td>stable front for 5 iterations</td></tr>
<tr><td colspan="2">gene pool</td><td colspan="4">≤ 5 genes per entity</td></tr>
<tr><td colspan="2" rowspan="2">elitism</td><td colspan="2">weight-based optimization</td><td colspan="2">Pareto optimization</td></tr>
<tr><td colspan="2">transfer rate 0.20</td><td colspan="2">Pareto front</td></tr>
<tr><td colspan="2">selection</td><td colspan="4">roulette wheel</td></tr>
<tr><td rowspan="6">crossover (apart from 10% of the elite)</td><td rowspan="5">uniform crossover at probability 0.90</td><td colspan="2">coverage</td><td colspan="2">exchange probability</td></tr>
<tr><td colspan="2">[0.00 ; 0.25]</td><td colspan="2">0.50</td></tr>
<tr><td colspan="2">]0.25 ; 0.50]</td><td colspan="2">0.40</td></tr>
<tr><td colspan="2">]0.50 ; 0.75]</td><td colspan="2">0.30</td></tr>
<tr><td colspan="2">]0.75 ; 0.90]</td><td colspan="2">0.20</td></tr>
<tr><td>cut & glue at probability 0.10</td><td colspan="4">]0.90 ; 1.00] 0.10 –</td></tr>
<tr><td rowspan="6">mutation</td><td rowspan="6">mutation probability per test case: 0.10</td><td rowspan="2">coverage</td><td colspan="4">mutation operators</td></tr>
<tr><td>a</td><td>d</td><td>r</td><td>m</td></tr>
<tr><td>[0.00 ; 0.25]</td><td>1.00</td><td>0.00</td><td>0.00</td><td>0.00</td></tr>
<tr><td>]0.25 ; 0.50]</td><td>0.80</td><td>0.10</td><td>0.05</td><td>0.05</td></tr>
<tr><td>]0.50 ; 0.75]</td><td>0.70</td><td>0.20</td><td>0.05</td><td>0.05</td></tr>
<tr><td>]0.75 ; 0.90]</td><td>0.60</td><td>0.30</td><td>0.05</td><td>0.05</td></tr>
<tr><td>]0.90 ; 1.00]</td><td>0.50</td><td>0.40</td><td>0.05</td><td>0.05</td></tr>
</table>

Table 2: Comparison of three test case generation techniques in terms of effort required.

coverage criterion	**no optimization**		**weight-based optimization** w_1= 0.99, w_2=0.01		**Pareto optimization** (from 1st Pareto front)	
	average size	std. deviation	average size	std. deviation	average size	std. deviation
all transitions	1.3	0.48	1.4	0.52	1.5	0.53
all events	25.2	1.99	24.2	1.55	17.7	1.64
all states	106.2	4.54	76.9	4.86	63.2	5.94
all state pairs	177.3	3.89	138.9	8.84	143.5	10.96

The CPN was modelled using CPN Tools (Jensen, Kristensen and Wells, 2007). The implemented algorithm is parameterized as shown in Table 1.

For the purpose of comparing the different test case generation strategies considered, an initial CPN marking involving 3 robots moving on 5 segments was chosen. The robots were assigned the following missions:

- Robot #1 is assigned the order of moving from segment 1 to segment 4;
- Robot #2 is assigned the order of moving from segment 2 to segment 5;

– Robot #3 is assigned the order of moving from segment 5 to segment 1.

The example involves 6 transitions, 72 events, 261 states and 470 state pairs. In accordance with the step-wise testing procedure proposed, the "all transitions", "all events", "all states" and "all state pairs" criteria were selected for the purpose of capturing context-free, local context-specific and global contextual behaviour. For each criterion 10 test case sets were generated, each achieving 100 % coverage w.r.t. the specific underlying criterion.

Table 2 shows the average size of the test case sets, as well as the standard deviation for each experiment.

The results were also compared with non-optimized test generation where test cases are constructed such that after reaching a state all potential events may occur at the same probability. Although non-optimized test generation revealed to be practicable for the weaker coverage criteria, the stronger ones required a considerably higher amount of test cases.

On the other hand, weight-based multi-objective optimization was able to help save up to 28% of test cases in comparison with non-optimized test generation. Though practicable for complex systems, it assumes that the objectives were prioritized in advance. This requirement may reveal as a drawback as the tester usually does not have any a priori evidence about the practical implications of this choice.

This drawback is overcome by Pareto optimization allowing the tester to take an a posteriori decision among a set of optimal candidates. This may be particularly helpful when 100% coverage would require a too high amount of test cases, such that the tester might prefer to opt for a slightly lower coverage involving considerably less test cases.

6 CONCLUSION

This article presented a CPN-based testing procedure for cooperating autonomous software-based agents. It relies on successive testing phases of increasing contextual scope.

The testing techniques developed are considered as particularly relevant for the purpose of verifying and validating the cooperative behaviour of safety-relevant controllers, each governing the behaviour of an agent, as they allow to observe multiple scenarios involving whole varieties of potential interactions.

For the purpose of generating optimized test case sets evolutionary techniques revealed to be particularly helpful in maximizing interaction coverage while minimizing test amount. Two optimized test generation procedures using genetic algorithms were implemented and applied to a CPN model of cooperating autonomous forklifts. Compared with non-optimized test case generation they proved to be beneficial in saving testing effort.

Furthermore, Pareto optimization allows for an a posteriori adaptation of target priorities based on the actual amount of test cases being required to achieve given coverage criteria.

On the whole, the approach presented in this article offers a systematic testing procedure for cooperative autonomous systems intended to define and to measure objective testing targets, as well as to achieve them by a minimum number of automatically generated test cases.

ACKNOWLEDGEMENTS

It is gratefully acknowledged that part of the work reported was sponsored by the German Federal Ministry of Education and Research BMBF (Bundesministerium für Bildung und Forschung) in cooperation with the European Union Research Programme ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems), project R3-COP (Resilient Reasoning Robotic Co-operating Systems).

REFERENCES

Broy, M. et al., 2005. Model-Based Testing of Reactive Systems - Advances Lectures. In *Lecture Notes in Computer Science*, vol. 3472. Springer.

Freitas, A., 2002. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer.

Genrich, H. J., Lautenbach, K, 1981. System Modelling with High-Level Petri Nets. In *Theoretical Computer Science*, vol. 13(1), pp. 109-136. Elsevier.

Jensen, K., Kristensen, L. M.; Wells, L., 2007. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3-4, pp. 213-254. Springer.

Jensen, K., Kristensen, L. M., 2009. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer.

Lill, R., Saglietti, F., 2012a. Model-based Testing of Autonomous Systems based on Coloured Petri Nets. In *ARCS 2012 Workshops Proceedings, Lecture Notes*

in Informatics, vol. 200, pp. 241-250. Gesellschaft für Informatik.

Lill, R., Saglietti, F., 2012b. Test Coverage Criteria for Autonomous Mobile Systems based on Coloured Petri Nets. In *FORMS/FORMAT 2012 Proceedings, 9th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems*, pp. 155-162. Institut für Verkehrssicherheit und Automatisierungstechnik, TU Braunschweig.

Micskei, Z. et al., 2012. Concept for Testing Robustness and Safety of the Context-Aware Behaviour of Autonomous Systems. In *Proc. of the 1st Int. Workshop on Trustworthy Multi-Agent Systems (TruMAS)*, KES-AMSTA 2012, pp. 504-513.

Milner, R. et al., 1997. *The Definition of Standard ML (Revised)*. MIT Press.

Mitchell, M., 1996. *An Introduction to Genetic Algorithms*. MIT Press.

Nguyen, C. D. et al., 2012. Evolutionary testing of autonomous software agents. In *Autonomous Agents and Multi-Agent Systems*, vol. 25(2), pp. 260-283. Springer.

Murata, T., 1989. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, vol. 77, no. 4. pp. 542-543. IEEE.

Pelánek, R., 2009: Fighting state space explosion: Review and evaluation; In *Lecture Notes in Computer Science*, vol. 5596, pp. 37-52. Springer.

Saglietti, F., Söhnlein, S., Lill, R., 2011. Evolution of Verification Techniques by Increasing Autonomy of Cooperating Agents, In *Autonomous Systems: Developments and Trends*, Studies in Computational Intelligence, vol. 391, pp. 353-362. Springer.

Utting, M., Legeard, B., 2007. *Practical Model-based Testing - a Tools Approach*. Elsevier.

Valmari, A., 1998. The state explosion problem. In *Lecture Notes in Computer Science*, vol. 1491, pp. 429–528. Springer.

Westergaard, M., Kristensen, L. M., 2009. The Access/CPN framework: a tool for interacting with the CPN Tools simulator. In *Applications and Theory of Petri Nets,* pp.313-322. Springer.

Zhu, H., He, X., 2002. A methodology of testing high-level Petri nets. In *Information and Software Technology*, Vol. 44, pp. 473-489. Elsevier.