Autonomous Systems: Developments and Trends (H. Unger, K. Kyamakya, J. Kacprzyk eds.)
Studies in Computational Intelligence, Volume 391, © Springer-Verlag 2011
The original publication is available at www.springerlink.com.

Evolution of Verification Techniques by Increasing Autonomy of Cooperating Agents

Francesca Saglietti, Sven Söhnlein, Raimar Lill

Chair of Software Engineering
University of Erlangen-Nuremberg, Erlangen (Germany)
{saglietti, soehnlein}@informatik.uni-erlangen.de

Abstract. As system parts are becoming increasingly decoupled, gaining at the same time in terms of local autonomy, this article elaborates on the effects this trend has on verification and validation techniques. Both qualitative approaches to fault detection and quantitative approaches to reliability assessment are analyzed in the light of their evolution to adapt to the increasing decentralization and autonomy of modern 'systems of systems'.

1. Introduction

The responsibility role assumed by software-based applications when controlling a variety of critical tasks is continuously increasing. While in previous times the limited functionality of such systems allowed for monolithic designs, their increasing complexity has led - over the past decades – to growing levels of modularity, ranging from simple well-structured programs to sophisticated component-based systems including pre-developed packages.

Evidently, the structure introduced by modularity strongly supports transparency and understandability by endorsing the principles of abstraction and of separation of concerns, in particular contributing to lean maintenance by favoring change management via highly cohesive modules (easing fault localization) and loosely coupled modules (reducing potential side effects arising during fault removal). On the other hand, the insertion of pre-developed components was shown to introduce new fault sources in case the original application context in which the components originate slightly differs from the future context addressed by the new system to be developed [5].

This trend towards decoupled software parts is further increasing, typically for systems whose functional scope is dynamically evolving with time, like

- critical services provided on internet platforms (*cloud computing*);
- controllers protecting different assets considered as essential for the functioning of a society (*critical infrastructures*), like energy generation, transmission

and distribution, telecommunication, water supply, food production and distribution, public health, transportation, financial and security services;
- mechanical, intelligent agents performing individual or common tasks (*cooperative robots*).

While the motivation leading to such decentralized, highly decoupled and autonomous sub-systems – namely, ease of system evolution over time as well as growth of time efficiency and service flexibility – clearly strengthens their attractiveness, the high dependability demands posed on such software applications render their verification even harder than is already the case for more modest functional scopes.

In fact, as the individual system parts evolve with time and behave at a high degree of autonomy, the multiplicity of their interplay increases at rapid pace and is extremely difficult to be systematically captured by testing. On the other hand, in order to verify acceptable behavior, an accurate preliminary analysis should identify the variety of potential scenarios involving the interaction of autonomous parts and to assess their adequacy by representative test cases.

The intention of the present contribution is to elaborate on this novel challenge posed to software reliability engineering by illustrating recent and ongoing work addressing the evolution of verification activities with increasing autonomy of system parts.

2. Autonomy

The ancient Greek root of the term '*autonomy*' (*auto* = self + *nomos* = law) reveals that its original meaning generally referred to entities providing themselves with their own laws. Depending on the underlying political or ethical context, this definition may allow for slightly diverse interpretations
- within a *political context*, it refers to the self-government of human populations, while
- in terms of *moral philosophy*, it refers to the moral responsibility of an individual for his / her actions.

In both cases, it involves the capacity of an individual entity (human, population, or technical system) to make a rational and informed decision. In spite of the idealistic content hidden behind this assumption, it is well-known to software engineers – as it is to politicians and philosophers – that full autonomy (i.e. completely decoupled individual decisions) cannot help achieving an overall prioritized target of a society or an application.

The opposite, namely central controllers fully dictating actions to agents, evidently severely limit the potential capabilities provided at local level, restricting both performance and flexibility.

Between both extremes, the appropriate degree of autonomy is determined by optimizing the trade-off between individual freedom and essential rules of co-existence and cooperation; in case of software-based systems the latter are mainly characterized by

- *synchronization* activities required to achieve a common target, as well as
- *coordination* activities required to avoid resp. resolve conflicts.

Therefore, the degree of autonomy may be considered as higher

- the less *synchronization* constraints are required for cooperative tasks, and
- the more *coordination* tasks (including conflict resolution) are carried out in a decentralized way.

Formally, a community Com of cooperating units may be represented (as inspired by [6]) as

Com = (Aut, Init, Goal, Control)

where

- **Aut** denotes a set of units;
- **Init** denotes an initial global environment;
- **Goal** denotes an overall goal or a set of terminal global environments;
- **Control** denotes a set of rules concerning future behavior of units assuming knowledge of its present global environment.

Each unit aut $\in$ Aut is described as

aut = (init, goal, control)

where

- **init** denotes an initial local environment;
- **goal** denotes an individual sub-goal (the task of the individual), or a set of local environments;
- **control** denotes a set of individual rules concerning its future behavior assuming knowledge of relevant portions of its local environment.

The degree of autonomy characterizing each individual aut within a community Aut is then reflected by the relation between the scope of central functionality taken over by the *central intelligence* Control(Aut) and the scope of functionalities controlled by the *local intelligence* control(aut) of the individual considered:

Autonomy (aut) = funct[control(aut)] / funct[Control(Aut)]

In quantitative terms, this relation may be evaluated by

- the ratio of *logical complexities*, or by
- the ratio of *frequencies of occurrence*.

3. Verification

From the perspective of verification, the major challenge posed by autonomous systems relates to their inherent low compositionality. As the behavior of an individual may depend and may have an impact on the behavior of its operative environment, understanding each single part of a system (like in case of a conventional controller and of its controlled units) is not sufficient any longer. The central principle of '*separation of concerns'* which supports classical verification approaches, is hardly applicable here. On the contrary, at system level new global properties (so-called '*emergent behavior*') may arise, not easily deducible by logical inference, nor by composition of local properties. Emergent behavior may involve

- *intended*, expected or desired properties, implicitly targeted by the overall goal to be achieved by cooperation, but also
- *unintended,* surprising or undesired behavior (e.g. cascading) due to unpredicted side effects (e.g. deadlocks, or even unsafe behavior).

3.1 Qualitative Verification by Structural Analysis

A major approach to verification is based on (static or dynamic) analysis of system structure (so-called white-box or grey-box verification). In case of *monolithic units* of moderate size this is achieved by modeling the control flow (possibly with data flow annotations) and to analyze the model for the purpose

- of identifying *anomalies* (e.g. undefined or unused variables, dead code), and
- of selecting test data capable of achieving pre-defined *code coverage* criteria (unit testing [8]).

Component-based systems are usually implemented by object-oriented components exchanging messages via method calls. Message passing usually takes place by synchronous communication (hand-shake); this reduces the possibilities of race conditions and of further side effects. An approach to capture component interaction by abstraction was developed in [9] by means of communicating state machines invoking each other by parameterized message passing, hereby triggering potential state transitions both in invoking and invoked components.

In such cases structural analysis will require (in addition to the already mentioned unit verification approaches at component level) particularly accurate analysis of the intra-modular structure for the purpose

- of identifying *interface inconsistencies* (e.g. units referring to deviating physical systems of reference [5]), and
- of verifying resp. falsifying relevant *system properties* (like safeness, or liveness) expressed in temporal logic by model checking;

- of selecting test data capable of achieving pre-defined *interaction coverage* criteria (e.g. state-based interaction coverage [9], or coupling coverage [1], [4]).

As already indicated above, the main challenge of truly autonomous systems lies in their lack of synchronicity, which is reduced to a minimum degree. Due to this fundamentally asynchronous behavior, the behavior of autonomous cooperative systems cannot be captured by mere communication of state machines exchanging messages for the purpose of triggering particular specific events.

As mentioned above, autonomy should involve as little communication as possible; in particular, it should not assume the existence of common external events aimed at synchronizing their parallelism. In order to allow for as much free concurrency as ideally possible, more powerful notations are required, as offered by Petri Nets. Existing verification activities include
- *static reachability analysis* techniques, aimed a.o. at the early identification of deadlocks or unsafe behavior;
- *dynamic analysis* techniques based on the simulation of different scenarios.

As mentioned in the introduction, an increasing trend towards systems-of-systems is observable, consisting of co-existing and co-operating, de-centralized, autonomous sub-systems originating from different contexts. Here, the verification of their appropriate interplay does not require a mere *integration* perspective, but rather a view of their *interoperability* ensuring
- fulfillment of *consistency* constraints on relations between decentralized values,
- constraints on reading and writing operations between sub-systems required for information *confidentiality* or data *integrity*,
- provision resp. release of *common resources* required for *availability* targets.

While integration testing of component-based systems could progress thanks to a number of novel and automatic test data generation techniques, interoperability testing of autonomous systems still poses crucial challenges to the systems engineering community. Some ongoing work aimed at providing a systematic procedure for capturing the multiplicity of interoperation scenarios to be covered by tests will be presented in chapter 4. The considerations presented above about verification approaches evolving with the degree of autonomy are summarized in Table 1.

evolution stage	**static analysis**	**dynamic analysis**
monolithic units	data flow, anomalies	unit testing, code coverage
component-based system	invariants, inconsistencies	integration testing, interaction coverage
autonomous systems	reachability, deadlocks	interoperability testing, interoperation coverage

Table 1: Evolution of structural analysis techniques

3.2 Quantitative Verification by Statistical Evidence

Similarly to
- *qualitative approaches* addressing fault detection, also
- *quantitative approaches* addressing reliability assessment

are evolving together with the increasing role of interplaying system parts.

For monolithic systems, reliability may be conservatively assessed by means of statistical tests based on
- a preliminary *hypothesis* claiming an upper bound on the unknown failure probability;
- *operationally representative and independent tests* carried out to such a length as to enable the acceptance or the rejection of the hypothesis at any given confidence level.

If too demanding, the required tests may be replaced by a collection of operational evidence. In such a case, operative experience is to be filtered such as to fulfill all assumptions concerning independent and operational representative test data and test runs, as required by statistical sampling theory. The application of this technique to a software-based gearbox controller is illustrated in [6].

As soon as the logical complexity and execution frequency of component interactions tend to dominate over system behavior, black-box statistical testing approaches as the one proposed above may suffer from bias in case the reliability estimation is based on one-sided evidence missing to capture fundamental interactions. In order to prevent this from occurring, the originally monolithic, black-box approach was extended to include a grey-box perspective by granting – in addition to the demands of statistical sampling – also coverage w.r.t. predefined coupling criteria as proposed in [1] and [4], giving raise to an interaction-driven variant of statistical testing [7].

Finally, in case of autonomous systems, an analogous extension is due, aimed at capturing a systematic and adequate choice of scenarios involving asynchronous behavior.

The evolution of quantitative techniques is summarized in Table 2.

evolution stage	**statistical testing strategy**
monolithic units	black-box statistical testing
component-based system	interaction-driven statistical testing
system of autonomous systems	scenario-driven statistical testing

Table 2: Evolution of statistical evidence techniques

4. Modeling Autonomous Systems by Colored Petri Nets

Several formal notations may be considered as modeling languages for cooperating autonomous systems. A comparative analysis in the light of their expressive power, scalability, analyzability and tool support revealed Colored Petri Nets ([2], [3]) as a notation among the most promising ones; therefore, it was selected for further investigations.

A Colored Petri Net CPN is a 9-Tuple $(P, T, A, \Sigma, V, C, G, E, I)$, where
- P denotes a finite set of so-called *places* $p \in P$;
- T denotes a finite set of so-called *transitions* $t \in T$, with $P \cap T = \emptyset$;
- $A \subseteq P \times T \cup T \times P$ denotes a set of directed *arcs* connecting either places with transitions or transitions with places;
- Σ denotes a finite set of non-empty, so-called *color sets*;
- V denotes a finite set of variables v with $type[v] \in \Sigma$ for all $v \in V$;
- $C: P \rightarrow \Sigma$ denotes the so-called *color function* attaching to each place $p \in P$ a color set C(p) consisting of so-called *colors*; $C(p)_{MS}$ denotes the multi-set over C(p) consisting of sets of colors, where each color member may occur more than once;
- $G: T \rightarrow EXP_V$ is a function attaching to each transition $t \in T$ a so-called *transition guard,* i.e. an expression G(t) over variables $v \in V$ with $type[G(t)] = Bool$;
- $E: A \rightarrow EXP_V$ is a function attaching to each arc $a \in A$ a so-called *arc expression* E(a) over variables $v \in V$, with $type[E(a)] \in C(p)_{MS}$, where p is a place connected with arc a; variables in outgoing arc expressions of a transition are also in ingoing arc expressions of the same transition;
- $I: P \rightarrow C(p)_{MS}$ denotes a function attaching to each place $p \in P$ a so-called *initial marking* M(p) of type $C(p)_{MS}$.

A transition $t \in T$ with input places p_i and input arcs a_i: $p_i \rightarrow t$, $i \in \{1,\ldots,k\}$ is enabled if and only if – after binding each variable $v_{ij} \in V$ in an input arc expression $E(a_i)$ with a corresponding color $c_{ij} \in C(p_i)$, $1 \leq j \leq r(i)$, $r(i) \leq Card(C(p_i))$:
- the value of G(t) w.r.t. the underlying variable binding is true and
- for each color c_{ij} the value of $E(a_i)$ is not higher than the number of tokens of same color in p_i.

After firing of transition t w.r.t. colors c_{ij}, a new marking is obtained from the previous marking by
- removing from each input place p_i as many tokens of color c_{ij} as resulting by evaluating the input expression $E(a_i)$ w.r.t. the firing-specific variable binding;
- adding to each output place p_o connected to t by an output arc a_o: $t \rightarrow p_o$ tokens in number and color as resulting by evaluating the output expression $E(a_o)$ w.r.t. the firing-specific variable binding.

4.1 Example: Robot Factory

The following example concerns the movement of an arbitrary number of robots (modeled by a color set RB) through a narrow lane consisting of an arbitrary number of segments (modeled by a color set SEGMENT). A central controller sends orders (r, s, scurrent) to a robot r providing it with a mission by indicating its target segment s and its initial position ‘scurrent’ ≤ s. Robots must achieve this target as autonomously as possible by

- using their optical sensors to decide whether the next segment is free; i.e. segments where no other robot or further obstacles are visible);
- if free, moving to the next segment;
- if not, trying again to look ahead;
- enabling an alert after a pre-defined number of cycles.

Though having been intentionally restricted to a relatively simple functionality (namely exclusively forward movement, no workaround of obstacles, no deterministic time-out), it is felt that the simple example shown in Figure 1 (edited by CPN Tools [3]) already provides valuable insight into the problem complexity by supporting a compact and scalable representation of a high variety of scenarios.

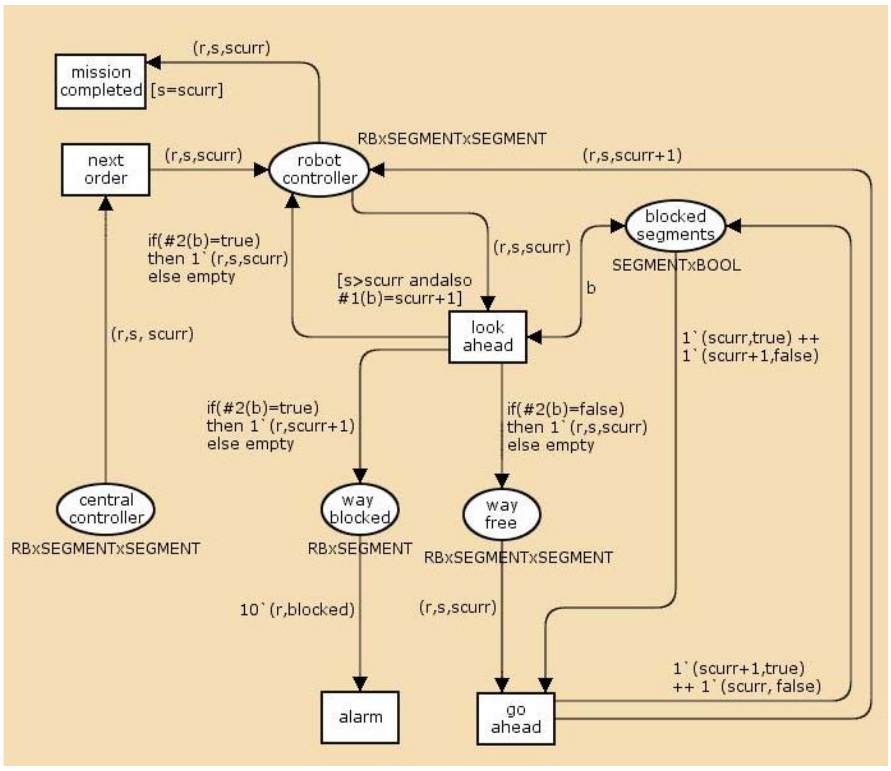

Figure 1: CPN Model of Robot Factory

In fact, the modeling notation chosen supports the following aspects:

- a compact representation of complex systems, consisting of cooperating autonomous entities, by providing the expressive power offered by token colors and transition constraints;
- high scalability, thanks to the possibility of extending the number of robots and / or segments by adding token colors, but without changing the model structure;
- apart from the initialization phase, robots do not need any global knowledge; on the contrary, each of them contributes to update the local knowledge stored in the marking of place 'blocked segments' by indicating the effects of its own movement.

For the purpose of testing CPNs, it is felt that the few existing approaches ([11], [12]) on testing coverage criteria must be extended in order to capture global reachability and scalability properties. Once novel, appropriate coverage metrics are defined on this basis, it is planned to apply multi-objective evolutionary techniques for the purpose of supporting the automatic test data generation tools for autonomous systems, as already successfully done for architectures involving tighter component coupling ([8], [9]).

Conclusion

This article proposed an analysis of classical and novel verification techniques evaluating their evolution in the light of the increasing level of autonomy of system parts. Both qualitative fault detection approaches and quantitative reliability assessment approaches require new extensions and adaptations to fit the growing degree of decentralization and autonomy of modern 'systems of systems'.

Acknowledgement. It is gratefully acknowledged that part of the work reported was sponsored by the European Union Research Programme ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems), project R3-COP (Robust & Safe Mobile Co-operative Autonomous Systems).

References

[1] R. T. Alexander, A. J. Offutt: Coupling-based Testing of O-O Programs; Journal of Universal Computer Science, vol. 10(4), 2004

[2] K. Jensen: Coloured Petri Nets: Basic Contents, Analysis and Practical Use; Springer-Verlag, 1997

[3] K. Jensen, L.M. Kristensen: Coloured Petri Nets - Modelling and Validation of Concurrent Systems; Springer-Verlag, 2009

[4] Z. Jin, A. J. Offutt: Coupling-based Criteria for Integration Testing; Software Testing, Verification & Reliability 8(3), 1998

[5] M. Jung, F. Saglietti: Supporting Component and Architectural Re-usage by Detection and Tolerance of Integration Faults, 9th IEEE International Symposium on High Assurance Systems Engineering (HASE '05), IEEE Computer Society, 2005

[6] H.-J. Kreowski: Modeling Iteracting Logistic Processes by Communities of Autonomous Systems, talk on activities of DFG-SFB 637 (Selbststeuerung logistischer Prozesse), Chinese-German Symposion, Braunschweig, 2010

[7] M. Meitner, F. Saglietti: Software Reliability Assessment based on Operational Representativeness and Interaction Coverage; 24th International Conference on Architecture of Computing Systems (ARCS 2011), Workshop Proceedings, VDE Verlag, 2011

[8] N. Oster, F. Saglietti: Automatic Test Data Generation by Multi-Objective Optimisation; Computer Safety, Reliability and Security (SAFECOMP 2006), Lecture Notes in Computer Science, Vol. LNCS 4166, Springer-Verlag, 2006

[9] F. Saglietti, F. Pinte: Automated Unit and Integration Testing for Component-based Software Systems; Dependability and Security for Ressource Constrained Embedded Systems (D&S4RCES), ACM Digital Library, 2010

[10] S. Söhnlein, F. Saglietti, F. Bitzer, M. Meitner, S. Baryschew: Software Reliability Assessment Based on the Evaluation of Operational Experience; Measurement, Modelling, Evaluation of Computing Systems, Dependability and Fault Tolerance (MBB & DFT 2010), Lecture Notes in Computer Science, Vol. LNCS 5987, Springer-Verlag, 2010

[11] H. Zhu, X. He: A Theory of Testing High-Level Petri Nets, 16th IFIP World Computer Congress, 2000

[12] H. Zhu, X. He: A methodology of testing high-Level Petri nets, Information and Software Technology, Vol. 44, 2002