A method for trustworthiness assessment based on fidelity in cyber and physical domains

Vincenzo De Florio[†] and Giuseppe Primiero[‡]

[†]: MOSAIC, University of Antwerp & iMinds, 2020 Antwerpen, Belgium
[‡]: Department of Computer Science, Middlesex University, London, UK

February 9, 2015

Abstract

We introduce a method for the assessment of trust for n-open systems based on a measurement of fidelity and present a prototypic implementation of a complaint architecture. We construct a MAPE loop which monitors the compliance between corresponding figures of interest in cyber- and physical domains; derive measures of the system's trustworthiness; and use them to plan and execute actions aiming at guaranteeing system safety and resilience. We conclude with a view on our future work.

1 Introduction

Fidelity of an open system can be interpreted as the compliance between corresponding figures of interest in two separate but communicating domains, see [9]. In cyber-physical systems, perfect fidelity means that actions in the physical domain have a well-defined and stable counterpart in the cyber domain, and vice-versa. This is an ideal state, as no concrete cyber-physical system may guarantee at all times a perfect correspondence between its domains of action. In practice, fidelity is affected by circumstances that let the system drift from the optimal case. Our stance is that, by observing the characteristics of said drifting, we may introduce a fine-grained characterization of the quality of system trustworthiness.

To this aim, we introduce a practical method for the assessment of trust based on the measurement of fidelity in computational systems, including cyber-physical ones. As a way to measure fidelity drifting we propose to adopt and extend the approach described in [7, 10]. We propose to make use of the accrued information to assess the characteristics of drifting in fidelity; we derive from it dynamic properties of both user and machine; evaluate it in terms of system's trustworthiness and use it to execute safety-assurance actions. This generates a trust-induced MAPE-loop.

The paper is structured as follows. In Sect. 2 we describe the conceptual model of system fidelity; in Sect. 3 the model for fidelity evaluation is implemented in an architecture for a cyber-physical system; finally, in Sect. 4 we build the link to trustworthiness evaluation. We conclude by drawing some observation and describing further lines of research.

2 Conceptual understanding of System Fidelity

In the present section we introduce a conceptual model of system fidelity, derived from the one presented in [9]. Our starting point is the concept of n-open systems (nOS), characterized by the following properties:

- nOS interact with one or more of the environments they are deployed in.

- nOS base their action on the ability to sense n classes of raw facts taking place in their deployment environments.

- nOS are able to construct and maintain n classes of internal representations of the raw facts, called *qualia*.

- *Qualia* are, to some extent, faithful, meaning that they timely reflect the dynamic variation of the corresponding class of raw facts.

We discuss here *fidelity* as a characterization of the above-mentioned faithfulness. More formally, given any $S \in n$OS, we consider n classes of raw facts, $[r]_i, 1 \leq i \leq n$, and n classes of binary operations, $[\dotplus]_i$, such that each of the couples

$$([r]_i, [\dotplus]_i) \tag{1}$$

constitutes an algebraic structure; for instance, when $[\dotplus]_i$ is a singleton, then $([r]_i, [\dotplus]_i)$ is a group. Likewise, for any $1 \leq i \leq n$, we call $[q]_i$ the class of *qualia* corresponding to $[r]_i$ and $[\oplus]_i$ the class of binary operations corresponding to $[\dotplus]_i$. Moreover, as we did for (1), we assume that $([q]_i, [\oplus]_i)$ is an algebraic structure. Then, for each $1 \leq i \leq n$, we consider the following function:

$$\Phi_i : [r]_i \to [q]_i, \tag{2}$$

mapping the qualia corresponding to any raw fact in $[r]_i$. We refer to the Φ_i functions as the *reflective maps* of some n-open system S. Reflective maps are assumed to be bijective functions, with Φ_i^{-1} being the *inverted reflective maps* of S associating the raw fact corresponding to each input quale.

We shall say that Φ_i expresses *perfect fidelity* between $([r]_i, [\dotplus]_i)$ and $([q]_i, [\oplus]_i)$ if and only if Φ_i preserves its algebraic structures (i.e., it is an isomorphism). More formally, for any couple of raw facts $(r_1, r_2) \in [r]_i \times [r]_i$ and for all $+ \in [\dotplus]_i$ and all $\cdot \in [\oplus]_i$:

$$\Phi_i(r_1 + r_2) = \Phi_i(r_1) \cdot \Phi_i(r_2).$$

Perfect fidelity may be better understood through an example. Let us assume that S is a cyber-physical system responsible for the operation of a mission critical hard-real-time service. An operator is responsible for the issuing of requests for service, which is done through a user interface (UI). A set of raw facts and prescribed behaviours pertaining to the physical environment are represented as "cyber-qualia" and "cyber-behaviours" stored in computer memories. Likewise, a set of "UI-qualia" and "UI-behaviours" are respectively rendered and operable through the UI. Perfect fidelity states that the correspondence between the physical, the cyber, and the UI domains is such that the prescribed behaviours as well as the referred raw facts and qualia are consistent on either of the involved domains. Thus certain operations and objects represented and rendered via the UI perfectly correspond to operations and objects encoded in S's computer components, which in turn perfectly correspond to physical actions having effects on physical entities.

Obviously, perfect fidelity only represents a reference point and can not be sustained and guaranteed at all times in real life. Imperfect fidelity is exemplified by the dreadful accidents experienced by the linear accelerator Therac-25 [21, 8]. An analysis of the causes of those accidents ascertained one of the reasons that led to some of the accidents was that the UI-qualia and the cyber-qualia did not match when the operator was typing at a very fast pace. As a result of such imperfect fidelity, the quantities represented on the screen of the Therac-25 did not correspond to the data stored in its memories—and, regrettably, to the amount of radiations supplied to the patients by the linear accelerator.

A slightly different and more practical definition of fidelity is given by a function $\phi_i, 1 \leq i \leq n$:

$$\phi_i : [r]_i \to [q]_i, \quad \text{such that} \ \ \forall + \in [\dotplus]_i, \forall \cdot \in [\oplus]_i : \phi_i(r_1 + r_2) = \phi_i(r_1) \cdot \phi_i(r_2) \cdot \Delta_i(t). \tag{3}$$

As for the Φ_i function, ϕ_i returns the qualia associated with the input raw facts. Contrarily to the Φ_i function, the ϕ_i does not preserve their algebraic structures unless the value of the error component $\Delta_i(t)$ is zero. The use of the lower-case "ϕ" is meant to suggest that ϕ_i represents a less-than-perfect version of Φ_i. The $\Delta_i(t)$ quantifies a drifting in time (represented by variable t) of the ability to create a trustworthy "internal" representation of an experienced raw fact.

A classic example of the manifestation of this drifting—as well as of its consequences—is given by the system failure that resulted in 28 US Army reservists being killed and 97 injured by a Scud missile on 25 February 1991, during the Gulf War. This was the single deadliest incident of the war for US soldiers,

and it was caused by the last Scud fired during the Gulf War [24]. The missile-defense system was an nOS that interacted with its environment through a number of context figures that included velocity and time. As discussed in [18], the cyber-quale corresponding to physical time was represented as the number of tenths of seconds from a reference epoch and stored in a 24-bit integer variable. Imprecisions in the conversion of said variable into a real number translated in an *unbound drifting of fidelity over time.* The more the Patriot missile-defense system operated without a reboot, the larger was the Δ_i pertaining to time and, as a consequence, the greater the discrepancy between the expected and the real position and velocity of the incoming Scud missile. An obvious workaround to the above unbound drifting is that of rebooting the system regularly so as to rejuvenate [17] the qualia management system and bring back the Δ_i to "safe" values. Although both problem and workaround were known at the time of the accident, no upper bound was known beyond which the resilience of the system would be affected. Common belief was that the unresilience threshold would never be reached in pratice. Regrettably, reality proved the trust on that belief to be misplaced. The Patriot missile that had to intercept the Scud never took off. As mentioned in the official analysis report [1],

> "at the time of the incident, the [system] had been operating continuously for over 100 hours". [Because of this long runtime,] "the [corresponding] inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud."

The above failure was due to the particular type of unbound fidelity drifting, monotonically increasing with time, experienced by system S. In [9], fidelity is classified in function of the type of drifting. Classes may include, e.g., the following cases:

- Hard-bound fidelity drifting, exemplified by hard-real-time nOS.

- Statistically-bound fidelity drifting—as typical of, e.g., soft real-time systems.

- Unbound fidelity drifting characterized by a "trend"—as in the case of the above mentioned missile interception system.

- Unbound fidelity drifting with a random trend.

The major benefit of expressing fidelity in terms of the characteristics of the experienced drifting is that the problem of assessing fidelity (and, therefore, resilience) translates in the problem of evaluating the properties of samples of the $\Delta_i(t)$ for dynamic systems.

In what follows, we propose an architecture and a prototypic system to evaluate systematically a system's fidelity drifting.

3 Janus' Architecture

In view of the above discussion, our approach to systematic evaluation of fidelity requires at least the following components:

- A sensory service, interfacing the deployment environments so as to register a number of "raw facts", namely variations in a number of environmental properties. Raw facts could refer, for instance, to variations in luminosity, temperature, or the amount of network bandwidth available between two endpoints.

- A uniform qualia service, providing consistent, timely and reliable access to cyber-qualia (computer-based representations of the raw facts).

- An application layer, providing a convenient means for fidelity assessment and reactive control.

In what follows we introduce the above components and a prototypic system compliant to the just sketched models, see Fig. 1.
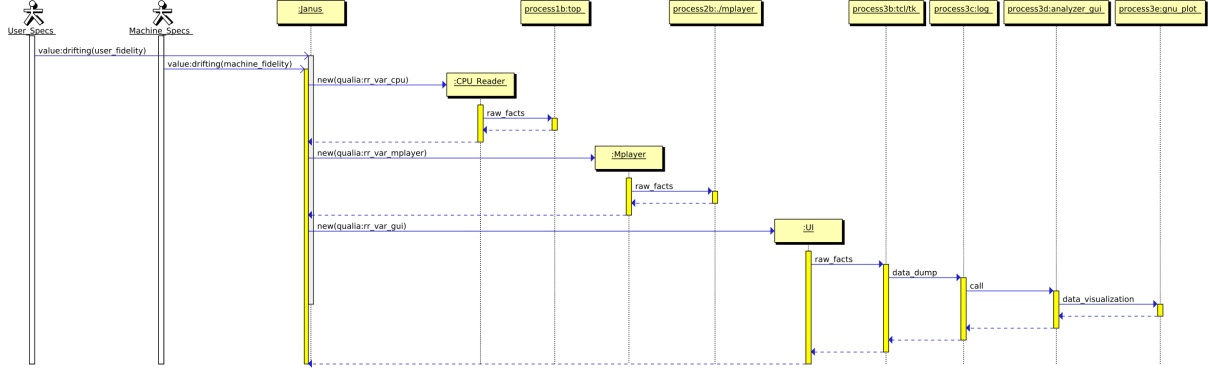
Figure 1: Sequence diagram of the Janus' components.

3.1 Sensory and Qualia Services

Our sensory and qualia service is based on so-called reflective and refractive variables (RRvars) [12, 11], a tool for the development of nOS in the C programming language. The idea behind RRvars is quite simple: a number of predefined variables provide access to the qualia of corresponding raw facts. Those variables are "volatile", meaning that their content is asynchronously and continuously updated by an associated thread. Thus for instance RRvar `int cpu` does not retain a constant value; rather, it is continuously updated by the `Cpu()` thread. Such thread ciclically retrieves the percentage of CPU currently in use and stores it in the memory cells associated to `cpu` as an integer number ranging between 0 (CPU fully available) and 100 (no CPU available). In the current implementation, which runs on Linux and Windows/Cygwin systems, `Cpu()` retrieves its raw facts by calling the `top` utility, which "provides a dynamic real-time view of a running system" [3]. This is referred to as `process1b` in Fig. 1.

A second and slightly more complex example is given by RRvar `int mplayer`, a variable updated by thread `Mplayer()`, referred to as `process2b` in Fig. 1. The latter component communicates with an instrumented MPlayer movie player [2] through a simple UDP client/server protocol. By reading the content of `mplayer` one is informed of the state of the MPlayer. See Fig. 3. Currently, the following integer values are used:

```
#define UDPMSG_STOP     1 // mplayer has finished playing a video
#define UDPMSG_SLOW     2 // mplayer is encountering problems while playing a video
#define UDPMSG_PAUSED   3 // mplayer has been paused
#define UDPMSG_START    4 // mplayer has been launched
#define UDPMSG_SIGNAL   5 // mplayer caught an exception and is about to exit abnormally.
```

A third case is given by RRvar `int ui`, updated by thread `Ui()`. This is a special case in that this RRvar represents a UI-qualia (see Sect. 2) reporting raw facts specific of an instance of a user interface. This is referred to as `process3b-e` in Fig. 1. Said user interface and the `Ui()` thread communicate transparently of the operator via the same mechanism presented for RRvar `mplayer`. The values returned in RRvar `int ui` represent usability raw facts derived by comparing the behaviours exercised by the current user with "reference behaviours" representing the expected behavioural patterns of a trustworthy operator. The method to derive these raw facts is described in [7, 15]. This method may be used to detect two classes of behavioral driftings:

- Gradual behavioural driftings due to, e.g., fatigue, stress, or the assumption of psychotropic substances.

- Sudden behavioural driftings caused, e.g., by an account takeover or other cybercriminal attacks.

```
/* File Janus.c
 * created/modified on Mit Feb  4 09:48:11 CET 2015
 */

#include "reflection.h"
#include "rcode.h"
#include "rrvars_init.h"

int main (int argc, char *argv[])
{
        int cpuold, mplayerold, uiold;
        RR_VARS

        RR_VAR_CPU
        RR_VAR_MPLAYER
        RR_VAR_UI

        printf("cpu == %d\nmplayer == %d\nui == %d\n", cpu, mplayer, ui);
        while (1) {
                sleep(1);
                if (cpuold != cpu)
                printf("cpu == %d\n", cpuold=cpu);
                if (mplayerold != mplayer)
                printf("mplayer == %d\n", mplayerold=mplayer);
                if (uiold != ui)
                printf("ui == %d\n", uiold=ui);
        }
}
#include "rrvars_end.c"
```

Figure 2: Typical structure of an RRvar client. Here three RRvars (cpu, mplayer, and ui) are declared and continuously displayed
.

3.2 Control Layer: Janus

Janus is the name of our exemplary RRvar client component.[1] The structure of `Janus` is the one typical of RRvar clients [11] and exemplified in Fig. 2. As can be seen from the picture, the RRvar metaphor makes it possible to quickly define nOS components based on the three classes of qualia presented above, see Fig. 4.

4 Fidelity as Trustworthiness

In this final section, we link the model of open systems offered in Sect. 2, and their architecture implemented in Sect. 3, to a model of trustworthiness evaluation. We use fidelity to assess the working conditions of an open system and to establish a metric of trustworthiness. Trust for security, management and reputation systems is gaining a lot of attention in the literature and it is typically accounted for as a first-order relation between (possibly autonomous) agents or system's modules, see e.g. [4, 6, 5, 16, 25]. Our account considers trust as a second order property characterizing cumulatively system's fidelity, where the latter is obtained as the dynamic variation of properties of the system's modules. This approach to second-order trust has been already used for information transmission evaluations ([23]), access-based control ([22]), and software management systems ([19]). In the present analysis, trustworthiness is used to plan reaction to malfunctioning and restoring of functionalities in cyber-physical systems. As the cases of Therac-25 and the Patriot-missile defence system show, unacknowledged drifting can be crucial in maximizing unjustified trust to dangerous levels. On the other hand, a metric evaluating minimal functionality thresholds can minimize unjustified mistrust, reducing confidence that the system *will* choke and eventually fail (antitrust). Early cases of low true-alarm rate or high false-alarm rate in automation do not need to set constants for future behaviour of the user. Similarly, criteria to compare intended and current behaviour are essential to allow mechanical assessment of user's inability, incompetence or threats. Fidelity and drifting can provide the systematic methodology and quantitative model to continually evaluate and experiment the system's vulnerability in the context of its operations,

---

[1]As for the system described in [13], the name of our component comes from mythical *Janus Bifrons*, the god of transitions, who had two faces and thus could observe and reason by considering two different "views" at the same times. Of the proposed etymologies of Janus, particularly intriguing here is the one proposed by Paul the Deacon [14]: hiantem, hiare, "to be open". Due to this fact one would be tempted to refer to Janus Bifrons here as to a *2-open system*.
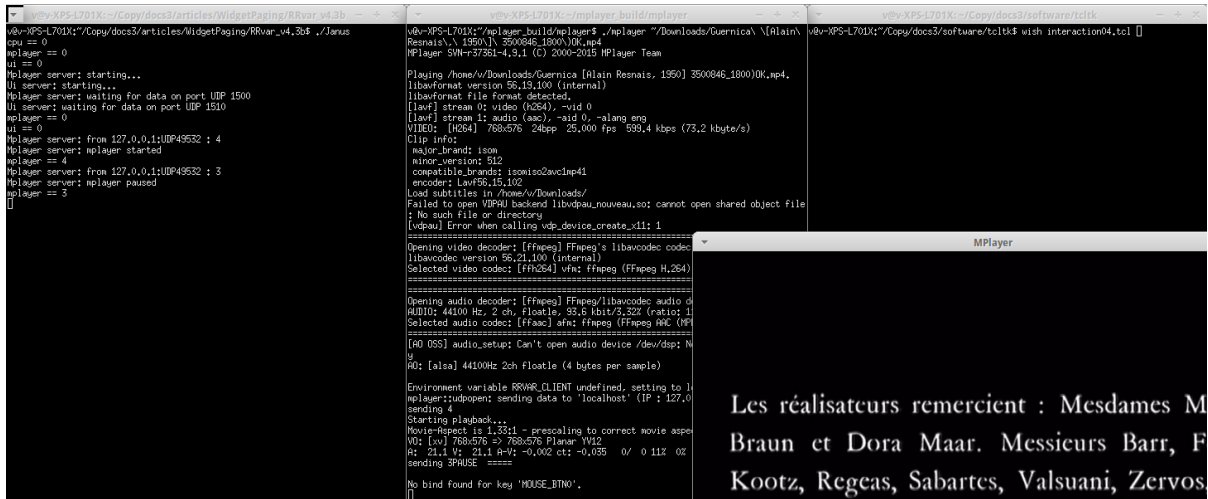
Figure 3: An instance of the MPlayer connects with our RRvar client informing of its state.
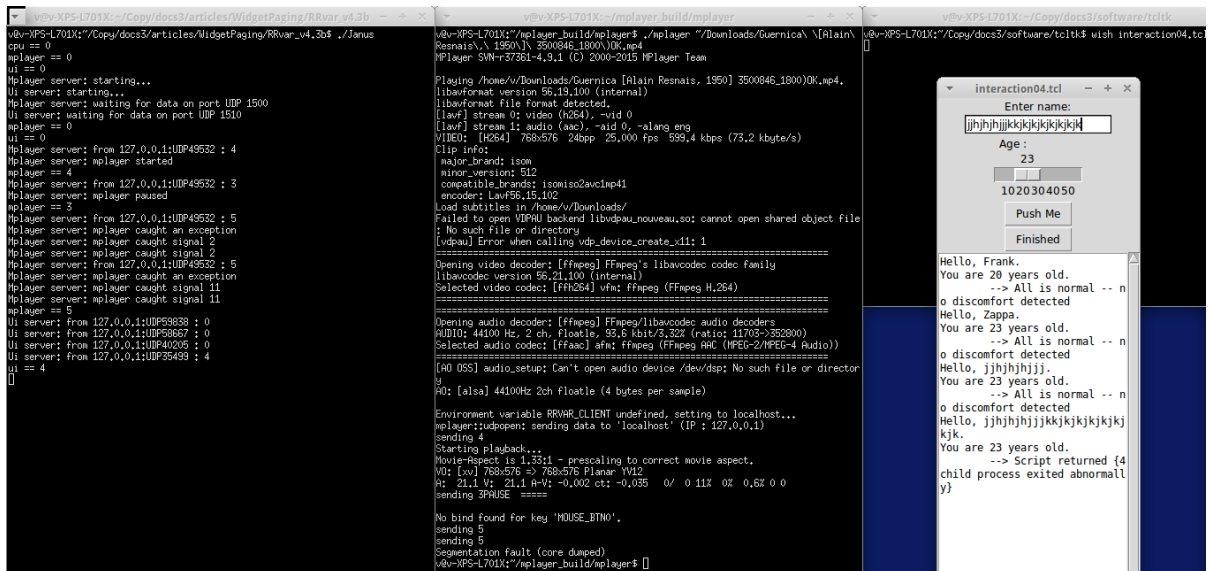

Figure 4: The RRvar client connects with both MPlayer and an exemplary user interface.
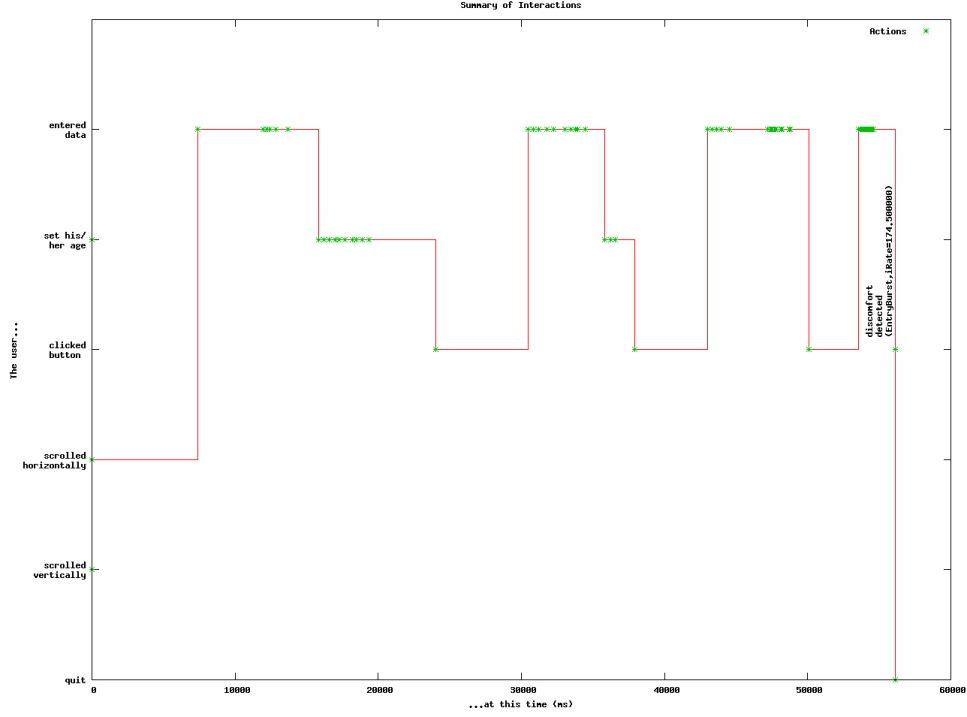
Figure 5: Log of the interaction between user and user interface. Between 50 and 60s a rapid burst of keystrokes is interpreted as an anomalous situation.

thus making the dichotomy conditional vs. unconditional trust (faith) working. Trust can be thought of as a measure of confidence about minimal drifting from fidelity for all the components, and hence that the overall behaviour of the system is sufficiently resilient. In the following, we reconstruct the process of fidelity monitoring, trustworthiness evaluation and operation execution in terms of a MAPE-loop designed for trust, see [20].

In the architecture presented in Sect. 3, the *Janus* assesses the behaviour of the system. This is parametrized in view of reflective variables for CPU consumption and a component's operations (on the machine side of the system), and for the user interface (on the user's side). Mapping of these variables values as raw facts and qualia provides a measure of system's fidelity:

- $\Phi_{CPU} : [r]_{CPU} \rightarrow [q]_{CPU}$, obtained by the mapping of input values from the *top* process to pre-selected parameters assigned to CPU-consumption behaviour;

- $\Phi_{component} : [r]_{component} \rightarrow [q]_{component}$, obtained by the mapping of input values from the executable's observable behaviour to pre-selected parameters assigned to its operations;

- $\Phi_{UI} : [r]_{UI} \rightarrow [q]_{UI}$, obtained by the mapping of input values from the user's observable behaviour to pre-selected parameters assigned to a standard or expected user's behaviour.

The system's component monitoring the classes $[r]_i$ of raw facts is called the *sensor layer*; similarly, we use *qualia layer* to refer to the component monitoring the classes $[q]_i$ of qualia. The combination of the sensory and representative layers constitutes the *Monitoring* component within our MAPE-loop. Fidelity is then approximated as the inversely proportional function of the drifting from appropriate mappings Φ_i. We shall refer to the value of user-based mappings as *user-defined fidelity* (Φ^U); correspondingly, we shall call *machine-defined fidelity* (Φ^M) the value based on mappings related to the machine behaviour. For the *Janus* introduced in Sect. 3,

$$\Phi^U = 1/\Delta(t)_{UI} \tag{4}$$

$$\Phi^M = 1/f(\Delta(t)_{CPU}, \Delta(t)_{exec}) \tag{5}$$

where f is some function, weighted according to domain-specific and user defined parameters. We refer to the set of values $\mathbf{\Phi}(t) = \{\Phi^U, \Phi^M\}$ as the content of our *Analysis* component, with the global value $\mathbf{\Phi}$ parametrized by time. As an example, consider the class of mappings Φ_{UI}, with a value of the sensor layer indicating e.g. quick typing and a value of the qualia layer returning a distress indication: in this case the fidelity layer reports an high value. As an example across distinct mappings, assume that the reflective variable for MPlayer indicates that the application is running slower, while the one for CPU monitoring indicates low usage value: this is expressed by a low fidelity value across the two classes in Φ^M. Analysing fidelity values across the distinct monitoring layers allows to obtain a cumulative evaluation. This value is monitored by the *apperception layer*. This layer is used to evaluate system trustworthiness as a global value of user-defined and machine-defined fidelity values. The next level is represented by the *Janus* feeding the content of the *apperception layer* into the *control layer*. This corresponds to the *Planning* component of our system. At this stage, system trustworthiness is matched to a resilience scale that identifies and automatically triggers actions aimed at preserving system safety or enabling ameliorating conditions. The latter part of the system is the *Execution* component, monitoring an *action layer*. Despite the fact that a resilience scale should be highly domain specific, a possible general model can be given in terms of four essential stages:

1. *Trustworthy System* identifies high levels of $\mathbf{\Phi}(t)$, inducing optimal, sustainable working conditions;

2. *Unstable System* identifies high-to-medium Φ^U and low Φ^M levels, inducing reconfigurable working conditions;

3. *Unsafe System* identifies high-to-medium Φ^M and low Φ^U levels, inducing alarm-rising working conditions;

4. *Untrustworthy System* identifies low-levels of $\mathbf{\Phi}(t)$, inducing inadvisable or below-safety working conditions.

The above analysis is summarized in the MAPE-loop in Fig. 4.

5 Conclusions

We have presented a prototypic model architecture for a trust-based MAPE-loop for cyber-physical systems. The trust evaluation is grounded on the assessment of fidelity drifting with respect to values representing ideal reference conditions for both user and machine. Although prototypic, we believe that our architecture proves the feasibility of our approach to trustworthiness assessment and paves the way towards future implementations. Our goal is to develop systems that are trustworthy in integrating quality-of-service and quality-of-experience, by optimising the relations between system-level, "microscopic" aspects and user-level, "macroscopic" ones. A further goal is to extend our architecture into that of a MAPE-K loop and apply machine learning methods such that systems based on our approach may systematically improve the match with the environments they interact with. A side line of research is the required development of formal logics for the modeling of the system trustworthiness assessment in a probabilistic epistemic setting.

References

[1] Anonymous. Patriot missile defense: Software problem led to system failure at dhahran, saudi arabia. Technical Report GAO/IMTEC-92-26, U.S. Government Accountability Office, US, General Accounting Office, Washington, D.C., 20548, 1992.

[2] Anonymous. Mplayer — the movie player, 2015. Retrieved on February 3, 2015 from www.mplayerhq.hu/design7/info.html.
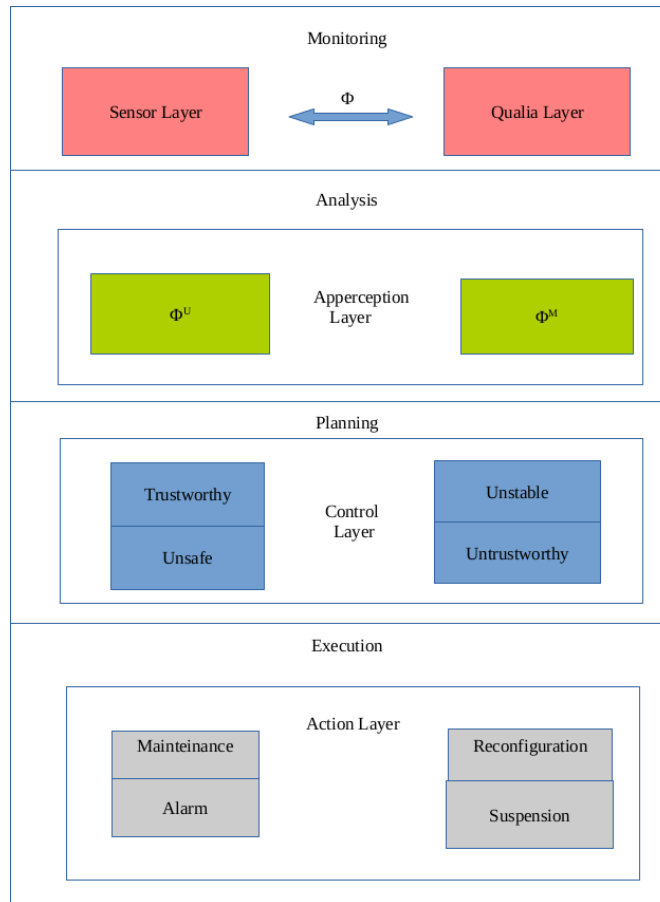
Figure 6: A MAPE-loop for System trustworthiness based on fidelity.

[3] Anonymous. Top - display linux processes, 2015. Manual page obtained through the command "man top" on Linux systems.

[4] Marco Carbone, Mogens Nielsen, and Vladimiro Sassone. A formal model for trust in dynamic networks. In *1st International Conference on Software Engineering and Formal Methods (SEFM 2003), 22-27 September 2003, Brisbane, Australia*, page 54. IEEE Computer Society, 2003.

[5] Jian Chang, Krishna K. Venkatasubramanian, Andrew G. West, Sampath Kannan, Boon Thau Loo, Oleg Sokolsky, and Insup Lee. AS-TRUST: A trust quantification scheme for autonomous systems in BGP. In Jonathan M. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, Angela Sasse, and Yolanta Beres, editors, *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings*, volume 6740 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2011.

[6] Stephen Clarke, Bruce Christianson, and Hannan Xiao. Trust*: Using local guarantees to extend the reach of trust. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols XVII, 17th International Workshop, Cambridge, UK, April 1-3, 2009. Revised Selected Papers*, volume 7028 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2009.

[7] V. De Florio and C. Blondia. Safety enhancement through situation-aware user interfaces. In *System Safety, incorporating the Cyber Security Conference 2012, 7th IET International Conference on*, pages 1–6, Oct 2012.

[8] Vincenzo De Florio. Software assumptions failure tolerance: Role, strategies, and visions. In Antonio Casimiro, Rogério de Lemos, and Cristina Gacek, editors, *Architecting Dependable Systems VII*, volume 6420 of *Lecture Notes in Computer Science*, pages 249–272. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-17245-8_11.

[9] Vincenzo De Florio. Antifragility = elasticity + resilience + machine learning. models and algorithms for open system fidelity. *Procedia Computer Science*, 32:834–841, 2014. 1st ANTIFRAGILE workshop (ANTIFRAGILE-2015), the 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014).

[10] Vincenzo De Florio and Chris Blondia. Reflective and refractive variables: A model for effective and maintainable adaptive-and-dependable software. In *Proc. of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)*, Lübeck, Germany, August 2007.

[11] Vincenzo De Florio and Chris Blondia. Reflective and refractive variables: A model for effective and maintainable adaptive-and-dependable software. In *Proceedings of the 33rd Euromicro Conference on Software Engineering and Advanced Applications (SEEA 2007), Software Process and Product Improvement track (SPPI)*, Lübeck, Germany, August 2007. IEEE Computer Society.

[12] Vincenzo De Florio and Chris Blondia. On the requirements of new software development. *International Journal of Business Intelligence and Data Mining*, 3(3), 2008.

[13] Vincenzo De Florio, Geert Deconinck, Mario Truyens, Wim Rosseel, and Rudy Lauwereins. A hypermedia distributed application for monitoring and fault-injection in embedded fault-tolerant parallel programs. In *Proc. of the 6th Euromicro Workshop on Parallel and Distributed Processing (Euro-PDP'98)*, pages 349–355, Madrid, Spain, January 1998. IEEE Comp. Soc. Press.

[14] Paulus Winfridus Diaconus. *Excerpta ex libris Pompeii Festi de significatione verborum*. W. M. Lindsay, 1930.

[15] Jens Van Duyse. A toolkit for the concurrent analysis and adaptation of graphical user interfaces. Master's thesis, Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, 2020 Antwerpen, Belgium, 2013. Promotor: Vincenzo De Florio.

[16] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *Commun. Surveys Tuts.*, 3(4):2–16, October 2000.

[17] Michael Grottke, Matias Jr. Rivalino, and Kishor S. Trivedi. The fundamentals of software aging. In *Proceedings of the 1st International Workshop on Software Aging and Rejuvenation, 19 International Symposium on Software Reliability Engineering*, 2008.

[18] Michael Grottke and Kishor S. Trivedi. Fighting bugs: Remove, retry, replicate, and rejuvenate. *IEEE Computer*, 40(2):107–109, 2007.

[19] Giuseppe Primiero Jaap Boender and Franco Raimondi. Minimizing transitive trust threats in software management systems. Technical report, Foundations of Computing Group, Middlesex University, 2015.

[20] Bart Jacob, Richard Lanyon-Hogg, Devaprasad K Nadgir, and Amr F Yassin. *A Practical Guide to the IBM Autonomic Computing Toolkit*. IBM Redbooks, 2004.

[21] Nancy Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[22] Giuseppe Primiero and Franco Raimondi. A typed natural deduction calculus to reason about secure trust. In Ali Miri, Urs Hengartner, Nen-Fu Huang, Audun Jøsang, and Joaquín García-Alfaro, editors, *2014 Twelfth Annual International Conference on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24, 2014*, pages 379–382. IEEE, 2014.

[23] Giuseppe Primiero and Mariarosaria Taddeo. A modal type theory for formalizing trusted communications. *J. Applied Logic*, 10(1):92–114, 2012.

[24] Eric Schmitt. After the war; army is blaming patriot's computer for failure to stop the dhahran scud. *New York Times*, May 1991.

[25] Zheng Yan and Christian Prehofer. Autonomic trust management for a component-based software system. *IEEE Transactions on Dependable and Secure Computing*, 8(6):810–823, 2011.