Towards Quality-Oriented Architecture: Integration in a Global Context

Maria Spichkova, Huai Liu, Heinz W. Schmidt
RMIT University
Melbourne, Australia
{maria.spichkova, huai.liu, heinz.schmidt}@rmit.edu.au

ABSTRACT

This paper introduces an architectural framework for developing systems of systems, where the development plants are geographically distributed across different countries. The focus of our ongoing work is on architectural sustainability, in the sense of cost-effective longevity and endurance, and on quality assurance from the perspectives of integration in a global context. The core of our framework are different levels of abstraction, where state-of-the-art industrial development process is extended by the level of remote virtual system representation. Each abstraction level is associated with a different level of context-dependent architecture as well as the corresponding testing approaches.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

Keywords

logical architecture, globally distributed systems, sustainability

1. CHALLENGES AND MOTIVATION

The key question in engineering and production is how to reach engineering goals in a way that minimises the overall effort. A large part of the answer to this question is in choice of an appropriate architecture, design process as well as appropriate abstraction levels during each design stage. In state-of-the-art industrial development, quality assurance is performed by extensive testing of generated code and of the real system which is physically present for testing. However, if the development is geographically distributed across different countries, states, and organisations, we need to consider the integration and interoperability of the system's elements in a global context. We suggest to have an additional phase in the development process: *virtual integration/interoperability testing*.

SAGRA '15, September 8, 2015, Dubrovnik, Croatia

An interesting question is whether remote cyber-physical integration/interoperability testing can or should have any influence on the elaboration of a logical architecture and its analysis. Can we take advantage of abstraction of information about the further development process? Indeed, can we even defer the decision of whether some integration and testing steps will be done remotely in the future? Otherwise, which architecture aspects should be taken into account already at the logical level if the development process includes remote integration and testing?

Let us discuss an example scenario based on the ideas of the virtual interoperability testing. In an industrial plant we require the integration/interoperability of $n+1$ bulky/heavy robots: one robot is of the type $AType$, the n other robots are of a different type $BType$. For each robot we have a number of features that can influence on the result of the test, e.g., a position of the robot (including relative position(s) of its grief(s) to its frame/trunk) and an action to perform. As additional features, we can see robots' relative positions to each other and the differences in configurations in the case when n robots of the same type perform similar movements and actions simultaneously. This implies that we need to test a huge number of possible configurations of the system to ensure its correct work. Which architectural configurations do we necessarily need to test? Which configurations could be excluded? How can we prioritise the tests, based on the logical architecture?

These questions are especially complicated if the production that is distributed over different locations, as this requires extensive simulation, testing and collaboration. In the case where components of a system are manufactured at different places, transporting from component development and production locations to integration and deployment sites can significantly increase the whole development costs as well as time. Integration can reveal additional work tasks and further transportation of the system's parts may be necessary. If a system's components are bulky or heavy, this may also delay optimisation and correction.

Given the high cost of physical integration and deployment as well as the huge number of possible features, it is necessary to apply a systematic approach for designing test suite that can cover most of the critical features and their combinations with a reasonable suite size. In our ongoing work, we make use of combinatorial testing [11], which aims to systematically design a covering array test suite which collectively represent all combinations of different factors for the system under test, such as input parameters, features, configurations, components, etc.

Contribution: We propose an *architecture-based combinatorial approach to integration/interoperability testing for systems in a global context*. On one hand, we apply the combinatorial testing technique to design a controllable number of tests that can reasonably cover various combinations of architectural features, and thus to detect possible failures that might be triggered by the interactions among features during the integration and deployment process. On the other hand, we make use of the architectural views in different abstraction levels to define the constraints among different system features, and in turn, to further improve the quality of test suite by automatically eliminating the invalid combinations of features. The tests generated by the approach are executed in a virtual environment, where the features are simulated rather than physically deployed in the field, to save the cost and remove any potential defects prior to the real integration and deployment.

Thus, we focus on the sustainability aspects of the architecture, in the sense of cost-effective longevity and endurance, i.e. we focus on *technical sustainability* [14]. However, we do not take into account another dimensions of sustainability such as social, environmental, etc. sustainability [7].

Outline: The rest of this paper is organised as follows. In Section 2, we introduce different architecture levels, which are the core of our framework. In Section 3, we discuss the general ideas of virtual integration/interoperability testing and then propose our combinatorial approach to the quality-oriented architecture. The related work is discussed in Section 4. Finally, in Section 5 we present our conclusions.

2. LEVELS OF ABSTRACTION

The aim of a software architecture is to design a draft of the solution for the problem described in the corresponding requirements specification [3, 6]. While modelling system architecture at each level of abstraction, we should analyse whether we can mark some system elements/properties as too concrete for the current specification layer and omit them for keeping the architecture more readable and manageable (especially in the sense of testing and verification). We also need to analyse which kind of system's aspects and assumptions at the level of logical architecture impact most on fidelity of the model with respect to the real system.

Even if the information is not important at the current level, it might be able to influence on the overall modelling result after some refinement steps, i.e. at more concrete levels that are nearer to the real system in the physical world. Therefore, when specifying system architecture we should make transparent all the decisions on abstraction in the model and track them explicitly – in the case of contradiction between the system architecture and the real system this allows us to find the problem more easily and faster.

We can say that any system S (and, respectively, its architecture) can be completely described by the corresponding set of its properties. At each level l of abstraction we can split it into two subsets: set $Refl^l(S)$ of the properties reflected at this level of abstraction, and set $Abstr^l(S)$ of properties from which we abstract at this level, knowingly or unknowingly.

With each refinement step we move some properties from the set *Abstr* to the set *Refl*, and in some sense the set *Abstr* represents the termination function for the system refinement process.

In general, the role of testing is not only to reveal/exclude bugs which arise in refinements (as in verification) but also to evaluate prototype (un)suitability which may arise from misunderstood requirements (as in validation). Moreover, in practice we view the abstraction levels as corresponding to stages in an imperfect process rather than views which are kept complementary and consistent.

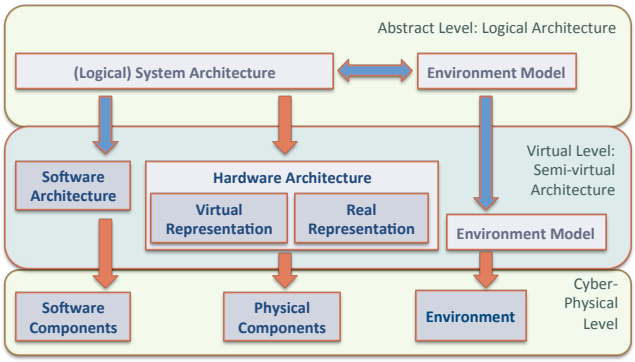

Figure 1: Generalised Development Methodology

The generalised methodology for the system of systems development in a global context is presented on Fig. 1:

- *Abstract Level*: We operate on the logical architecture of the system and an abstract model of the environment only. In a global context this also means that at this level we abstract from the information on which location each component of our logical architecture should be developed and deployed later.
  Quality assurance: We check the interoperability between logical components of our architecture (e.g., between abstract models of robots);

- *Virtual Level:* We distinguish software and hardware architectures and operate on both virtual and real representations of the hardware components. In a global context this means that we deal with a *semi-virtual architecture*, which takes into account where the software and hardware components should be developed and deployed later.
  Quality assurance: We check the interoperability between virtual and real systems. For this purposes we suggest to use a facility similar to the Virtual Interoperability Testing Laboratory (VITELab, cf. [2, 20]), where the interoperability simulation and testing are performed early and remotely, e.g., while cyber-physical components are in the prototyping stage. Individual components (e.g., robots, manufacturing cells), are connected in a suitable virtual environment, without being deployed at the same place physically. The aim of a VITELab approach is to reduce costs associated with interoperability testing at integration and commissioning phases in software-intensive automation application;

- *Cyber-Physical Level:* We operate on real system components.
  Quality assurance: In the case of testing, we check the interoperability between real systems that are physically present for testing, as in the state-of-the-art industrial development.

The proposed methodology is conform to the ideas of *Virtual Commissioning* technology [5, 12], which promises a more efficient handling of the complexity in assembly systems. We suggest to have three main meta-levels of abstraction:

At some level we need to switch from the pure abstract (logical) representation of the system to a cyber-physical one, but during a number of refinement steps we test (and refine) the system or component using a virtual environment, and then continue with testing in a real environment.

A crucial question for a quality-oriented architecture in a global context is which features we need to check at which level of abstraction. Testing as well as verification at the concrete level is more expensive than on an abstract one, especially if some corresponding corrections within the system are necessary. Thus, it makes more sense to have more intensive testing at logical level to reduce the overall size of test suite for the next levels as much as possible.

We suggest a "pessimistic" approach, which allows to have false positives at abstract levels (they should be corrected due refinement steps), but forces to build the model in the way to exclude false negatives. For example, if our tests at the logical levels show that

(i) there is no collision between robots A and B, when A and B execute the actions Act_1^A and Act_1^B in the positions Pos_1^A and Pos_1^B respectively, but

(ii) a collision between these robots is possible when, A and B execute the actions Act_2^A and Act_2^B in the positions Pos_2^A and Pos_2^B,

this means that the configuration (i) is safe and we don't need to check it further, but the configuration (ii) should be carefully checked at the virtual interoperability level. This solution is beneficial because it allows better prioritisation of tests by focusing on the potentially critical configuration.

3. TESTING FRAMEWORK

Our framework is composed of two main components: a virtual environment for testing and an architecture-based combinatorial approach to test suite design. Both components are discussed in this section.

3.1 Cyber-Virtual Level of Testing

Let us now come back to the robot scenario mentioned in Section 1. Assume that the robots of the type $AType$ are assembled in location L_A, while the robots of type $BType$ are assembled in a different location L_B. If n robots of the same type perform similar movements and actions simultaneously, we can simulate their behaviour using a single real robot.

Thus, we can simulate robots of type $BType$ by one real robot B, its actuator information will be replicated to obtain n virtual models $B_1, \ldots, B_n$, and its sensor information will be extended by the composition of the modelled sensor information from $B_1, \ldots, B_n$. The sensor information of the robot A will be a composition of the real sensor data and the sensor data modelled according to the actions of $B_1, \ldots, B_n$.

To check the interoperability of the robot A and n robots of the type $BType$ at the level of the semi-virtual architecture, we need only two real robots: a robot A and a robot B (cf. Fig. 2). Moreover, they could be located in L_A and L_B respectively, because the simulator and visualisation facility may take the role of a physical medium between them, allowing to ignore the real distance between robots and also providing a visualisation of the test and simulation not only at L_A and L_B, but also on the third place L_C, where the corresponding laboratory is located.

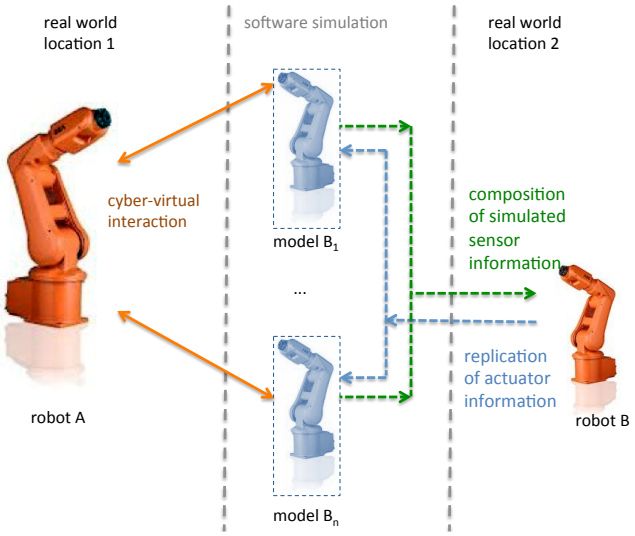

Figure 2: Cyber-virtual communication

Successful testing and simulation could significantly reduce the well-documented costs arising from discovery of design faults after implementation. Our models and their visualisation can give us the possibility

- to reveal a number of problems and inconsistencies on the early stage of system development;
- to verify important system's properties before the real system is built and integrated; and
- to identify possible weak points in the system (such as some timing properties, feature interactions, component dependencies) which we should focus on, during the testing phase.

3.2 Architecture-Based Combinatorial Testing

Even though the testing cost can be significantly reduced by using the virtual integration/interoperability testing facilities, the huge number of possible tests still largely affects the efficiency and effectiveness of testing. At the levels of logical and semi-virtual architecture, we mainly test the interoperability between virtual systems.

Let us look the following example, which is a simplification of the example above. Suppose that we only test two virtual robots, A and B, of types $AType$ and $BType$ respectively. Each robot can only take one action. We would have the following features for testing (where nA, mA, nB and mB are integers larger than 1, and x and y are the two largest numbers among them).

- Position of A: $Pos_1^A, Pos_2^A, \ldots, Pos_{nA}^A$.
- Action of A: $Act_1^A, Act_2^A, \ldots, Act_{mA}^A$.
- Position of B: $Pos_1^B, Pos_2^B, \ldots, Pos_{nB}^B$.
- Action of B: $Act_1^B, Act_2^B, \ldots, Act_{mB}^B$.

One possible test (i.e., a possible combination of features) is A taking Act_1^A at Pos_1^A with B taking Act_2^B at Pos_2^B. If there were no constraint among features, we could have a total of $nA \times mA \times nB \times mB$ possible combinations of features. However, we can significantly reduce the number of tests with combinatorial testing: there would be only $x \times y$ tests if we consider the 2-way combinations.

In the real-life testing, we have the situation where there are more than 2 components to test together, each component may take not just a single action but series of actions, and there are more features to test, other than positions and actions, the number of all possible combinations would be increased exponentially. Thus, the usage of combinatorial approach would become more critical for a controllable testing.

Furthermore, the different architecture levels will in turn help improve the combinatorial testing technique, if the abstract architectures contain innate properties for the systems under test. These properties allow us work out the constraints among different architectural features. One typical example for the constraint is that two robots cannot take the same position, that is, the combination of A at Pos_1^A with B at Pos_1^B is invalid and thus should not be used in the testing.

In the proposed approach, we introduce more well-defined constraints, such as "a certain action cannot be taken at a certain position", "Act_i^A cannot be taken simultaneously with Act_j^B", etc. An advantage of using these models is the high degree of automation, so it is natural that the constraint identification will easily be automatically conducted. It will also be easy for apply these architecture-level constraints into the combinatorial testing for the automated construction of a suite of valid tests.

4. DISCUSSION AND RELATED WORK

Model-driven architecture of embedded systems slowly becomes state-of-the-art of the development. A recent case study done by EIT ICT Labs [16] within the automotive domain (interviewees: 187 engineers and developers from 14 different countries) shows that (*i*) 97% of the participants apply functional modeling; (*ii*) 95% of the participants generate code from these models; (*iii*) more than 40% of the participants utilise these models for verification/validation within the development process. Nevertheless, many approaches on mechatronic/cyber-physical systems omit an abstract logical level of the system representation and lose the advantages of the abstract representation.

The work presented in [23] defines an extensive support to the components communication and time requirements, while the model discussed in [9] proposes a complete model of the processes with communication. In traditional development of embedded systems e.g., [1], the system is usually separated into software and hardware parts as early as possible in the development process. This does not always benefit the system development. When using the level of an abstract logical architecture the difference in the nature of components does not necessarily play an important role.

Some researchers [15, 18] suggested using a platform-independent architectural design in the early stages of system development. The approach presented by Sapienza et al. [15] introduces the idea of pushing hardware- and software-dependent design as late as possible. however, the question of the current practical and fundamental limitations of logical modelling in comparison to cyber-physical testing, is not completely answered. In comparison to [15], the focus of [18] is on reutilisation and generalisation of the software development methodologies based on the case studies supported by DENSO Corporation and Robert Bosch GmbH. However, the question, how deep we can go on the architecturing of cyber-physical systems at the logical level is still open in both approaches. In our work, we extend these ideas by integrating quality-oriented aspects into the architectural levels, which increases the architectural sustainability.

Gürbüz et al. proposed in [8] to apply the idea of safety perspective to ensure that the safety concern is properly addressed in the architecture views. The authors suggested to consider architectural tactics as possible solutions when the architecture does not exhibit the required quality properties addressed by the perspective. This approach can be embedded at each architectural level of our framework, to assist safety engineers and architects to identify the required quality properties.

Another promising approach that can be applied on top of our framework, was presented by Caracciolo et al. in [4]: a suitable specification of quality requirements helps to reduce further the cost of testing.

Penzenstadler et al. [13] presents a literature review on sustainability in software engineering, aiming to provide an overview of different aspects of sustainability.

A systematic review of the architecture-level metrics for the sustainability (in the sense of cost-effective longevity and endurance, i.e. the *technical sustainability*) of software architectures is presented by Koziolek in [10]. Koziolek reviewed the suitability of existing methods for sustainability analysis, and elaborated on this basis a list of more than 40 architecture-level software metrics potentially useful in the context of sustainability evaluation. Trough this elaboration, he also identified a need for more empirical studies on architecture evaluation.

5. CONCLUSIONS

In this paper, we presented our ongoing work on virtual interoperability testing, where the development process is extended by an additional level of abstraction – the level of remote virtual system representation. We further propose that in the virtual testing environment, we should use an architecture-based systematic approach to design test suites that are not only effective in detecting various failures, but also efficient in saving testing cost and time.

We investigate how to apply the traditional combinatorial testing technique into our integration/interoperability testing of cyber-physical systems in the global context. We also aiming on increasing of the readability and understandability of tests, to conform with the ideas of human-oriented software development, cf. [21, 17].

Our approach is to integrate the models at different architecture levels and the combinatorial technique. The architecture-based combinatorial technique will help us design test suite by combining various features at abstract levels; while the abstract logical architecture will provide an automatic mechanism for identifying the constraints between features and thus help improve the quality of test suite by eliminating the invalid tests.

Even after the systems are physically integrated and deployed, the well-defined tests are still very useful for the remote testing and monitoring in the real physical environ-

ment. Such an approach can significantly reduce the testing cost and ensure most of features and their interactions have been tested before the physical integration and deployment process.

Future work: One direction of our future work is to study how to extend testing with verification. In contrast to testing, *verification* delivers a correctness proof for critical properties, but requires significant effort, especially if we refer to verification of the code, which is typically more complex than verification at the model level. If we combine our current approach with modeling and verification tool chains, e.g. [19], this would allow assuring the quality of cyber-physical systems in the most effective way.

Another direction is to perform cloud-based testing for all abstraction levels of our framework, e.g. using Chiminey platform [24], which was created as part of the Bioscience Data Platform project [22].

6. REFERENCES

[1] A. Berger. *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. CMP Books, 2002.

[2] J. O. Blech, M. Spichkova, I. Peake, and H. Schmidt. Cyber-virtual systems: Simulation, validation & visualization. In *9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2014)*, 2014.

[3] P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Müller, B. Penzenstadler, K. Pohl, and T. Weyer. Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science - Research and Development*, 29(1):21–43, 2014.

[4] A. Caracciolo, M. Lungu, and O. Nierstrasz. How do software architects specify and validate quality requirements? In P. Avgeriou and U. Zdun, editors, *Software Architecture*, volume 8627 of *Lecture Notes in Computer Science*, pages 374–389. Springer, 2014.

[5] R. Drath, P. Weber, and N. Mauser. An evolutionary approach for the industrial introduction of virtual commissioning. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 5–8, 2008.

[6] R. Ferrari and N. Madhavji. The impact of requirements knowledge and experience on software architecting: An empirical study. In *The Working IEEE/IFIP Conference on Software Architecture (WICSA '07)*, pages 16–16, 2007.

[7] R. Goodland. *Encyclopedia of global environmental change, chapter Sustainability: Human, social, economic and environmental*. Wiley & Sons, 2002.

[8] H. Gürbüz, B. Tekinerdogan, and N. Pala Er. Safety perspective for supporting architectural design of safety-critical systems. In P. Avgeriou and U. Zdun, editors, *Software Architecture*, volume 8627 of *Lecture Notes in Computer Science*, pages 365–373. Springer, 2014.

[9] T. Hadlich, C. Diedrich, K. Eckert, T. Frank, A. Fay, and B. Vogel-Heuser. Common communication model for distributed automation systems. In *9th IEEE International Conference on Industrial Informatics*, IEEE INDIN, 2011.

[10] H. Koziolek. Sustainability evaluation of software architectures: A systematic review. In *Proceedings of the Joint ACM SIGSOFT Conference: QoSA and ACM SIGSOFT Symposium*, QoSA-ISARCS '11, pages 3–12. ACM, 2011.

[11] R. Kuhn, R. Kacker, Y. Lei, and J. Hunter. Combinatorial software testing. *IEEE Computer*, 42(8):94–96, 2011.

[12] S. Makris, G. Michalos, and G. Chryssolouris. Virtual commissioning of an assembly cell with cooperating robots. *Advances in Decision Sciences*, 2012, 2012.

[13] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch. Sustainability in software engineering: A systematic literature review. In *Evaluation Assessment in Software Engineering (EASE 2012), 16th International Conference on*, pages 32–41, 2012.

[14] B. Penzenstadler and H. Femmer. A generic model for sustainability with process- and product-specific instances. In *Proceedings of the 2013 Workshop on Green in/by Software Engineering*, GIBSE '13, pages 3–8. ACM, 2013.

[15] G. Sapienza, I. Crnkovic, and T. Seceleanu. Towards a methodology for hardware and software design separation in embedded systems. In *Proc. of the ICSEA*, pages 557–562. IARIA, 2012.

[16] B. Schätz. The role of models in engineering of cyber-physical systems – challenges and possibilities. In *CPS20 Experts Workshop. CPSWeek 2014*, 2014.

[17] M. Spichkova. Design of formal languages and interfaces: "formal" does not mean "unreadable". In *Emerging Research and Trends in Interactivity and the Human-Computer Interface*. IGI Global, 2013.

[18] M. Spichkova and A. Campetelli. Towards system development methodologies: From software to cyber-physical domain. In *Proc. of the FTSCS*, 2012.

[19] M. Spichkova, F. Hoelzl, and D. Trachtenherz. Verified system development with the autofocus tool chain. In *2nd Workshop on Formal Methods in the Development of Software*, WS-FMDS, 2012.

[20] M. Spichkova, H. Schmidt, and I. Peake. From abstract modelling to remote cyber-physical integration/interoperability testing. In *Improving Systems and Software Engineering Conference*, 2013.

[21] M. Spichkova, X. Zhu, and D. Mou. Do we really need to write documentation for a system? In *International Conference on Model-Driven Engineering and Software Development (MODELSWARD'13)*, 2013.

[22] The National eResearch Collaboration Tools and Resources (NeCTAR) Project. `http://www.nectar.org.au/`.

[23] B. Vogel-Heuser, F. S., T. Werner, and C. Diedrich. Modeling network architecture and time behavior of distributed control systems in industrial plant. In *37th Annual Conference of the IEEE Industrial Electronics Society*, IECON, 2011.

[24] I. Yusuf, I. Thomas, M. Spichkova, S. Androulakis, G. Meyer, D. Drumm, G. Opletal, S. Russo, A. Buckle, and H. Schmidt. Chiminey: Reliable computing and data management platform in the cloud. In *Proceedings of the 37th International Conference on Software Engineering (to appear)*, ICSE 2015. IEEE, 2015.