EAGLE: Engineering softwAre in the ubiquitous Globe by Leveraging uncErtainty

Marco Autili, Vittorio Cortellessa, Davide Di Ruscio, Paola Inverardi,
Patrizio Pelliccione, Massimo Tivoli
Università dell'Aquila, Dipartimento di Informatica, Via Vetoio, L'Aquila, Italy
{marco.autili, vittorio.cortellessa, davide.diruscio, paola.inverardi, patrizio.pelliccione,
massimo.tivoli}@univaq.it

ABSTRACT

In the next future we will be surrounded by a virtually infinite number of software applications that provide computational software resources in the open Globe. Users will be keen on producing their own piece of software, by also reusing existing software, to better satisfy their needs, therefore with a goal oriented, opportunistic use in mind. The produced software will need to be able to evolve, react and adapt to a continuously changing environment, while guaranteeing dependability. The strongest adversary to this view is the lack of knowledge on the software's structure, behavior, and execution context. Despite the possibility to extract observational models from existing software, a producer will always operate with software artifacts that exhibit a **degree of uncertainty** in terms of their functional and non functional characteristics. We believe that uncertainty can only be controlled by making it explicit and by using it to drive the production process itself. In this paper, we introduce a novel paradigm of software production process that **explores** available software and assesses its degree of uncertainty in relation to the opportunistic goal G, assists the producer in creating the appropriate **integration** means towards G, and **validates** the quality of the integrated system with respect to G and the current context.

Categories and Subject Descriptors

D.2 [**Software Engineering**]; D.2.10 [**Software Engineering**]: Design—*Methodologies*

General Terms

Design, Theory, Verification

Keywords

Ubiquitous computing, uncertainty

1. INTRODUCTION

In the next future we will be increasingly surrounded by a virtually infinite number of software applications that provide computational software resources in the open Globe. This situation radically changes the way software will be produced and used: (i) it lifts the user from the passive role of consumer to the active role of software producer of new functionalities out of the existing ones to satisfy user's needs; (ii) it shifts the focus of software production from domain specific software applications to software integrator systems that should ease the collaboration of existing software for the realization of new functionalities; (iii) the produced software will be inherently dynamic since it needs to operate in a continuously changing environment and must be able to evolve and quickly adapt to different types of changes, even unanticipated, while guaranteeing the dependability today's users expect.

The first characteristic implies a *goal oriented*, *opportunistic use* of the software being integrated, i.e., the producer will only use a subset of the available functionalities. The second one implies the need to *extract suitable observational models* from existing and reusable pieces of software, and devise appropriate *integration* means (architectures, connectors, integration patterns). The third one implies the need to cope with *dynamicity* in the integration process to guarantee both functionalities and dependability of the resulting application through (on-the-fly) *validation*.

This scenario requires rethinking the notion of software life cycle for which the distinction between development time and execution time stages is not meaningful anymore, e.g., PLASTIC[1] and SMSCom [20]. Recent approaches recognize the need to produce, manage and maintain software models all along the software's life time in order to assist the realization and validation of system's adaptations while the system is in execution [8, 25]. EAGLE (Engineering softwAre in the ubiquitous Globe by Leveraging uncErtainty) builds on this model-based software production paradigm but focuses its attention on the distinctive element of the above described scenario, namely the inherent incompleteness of information about requirements, execution context, and existing software.

The paper is organized as follows: Section 2 aims to answer the questions: "*What is the new idea?*" and "*Why is it new?*". Section 3 aims to answer the question: "*What are the most related works by the same authors and by others?*". Section 4 introduces the methodology for achieving the objectives of EAGLE. Finally, Section 5 concludes by also aiming to answer the question "*What feedback do the authors expect from the forum?*".

2. EAGLE NOVELTY

The big challenge underlying the above envisioned scenario is that abundance of available software induces a lack of knowledge on the software. A software producer will less and less know the

[1]FP6 IST EU PLASTIC project, `http://www.ist-plastic.org/`.

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.

precise behaviour of a third party software, nevertheless she will use it to build her own application. This means that the producer will in general operate in an environment in which the software artefacts exhibit a **degree of uncertainty** in terms of their functional and non functional characteristics. Indeed, in the software domain we see a flourishing of tools and methods to elicit approximated behavioural models of running systems, as well as context models. This very same problem recognized in the software engineering domain [19] is faced in many other computer science domain, e.g., exploratory search [36] and search computing [12], as well as software risk management [10], economics and other social domains [28].

In order to face this problem and provide a producer with a supporting framework to engineer the future software applications we rely on an apparent paradox. The uncertainty, due to the usage of software whose characteristics are not completely known, can only be controlled by making it explicit and by using it to drive the production process itself. To support this new idea we borrow the following definition of uncertainty from Galbraith [18] who, accordingly with [28], "*defines uncertainty as the difference between the amount of information required to perform a task and the amount of information already possessed*". This calls for a novel **explore-integrate-validate** production process that (i) explores available software and makes explicit the degree of uncertainty associated with it in relation to an opportunistic goal G, (ii) assists the producer in creating the appropriate integration means towards G, and (iii) validates the integrated system to assess its quality with respect to G and the current context.

The goal of the EAGLE approach is to provide an integrated model-based framework for the engineering of dependable and adaptable software for ubiquitous environments by explicitly dealing with uncertainty. This framework should support a future engineering production process of dependable open-globe software, i.e., goal-oriented software systems that are opportunistically created by integrating under uncertainty existing software and that are dynamically evolving within a perpetually changing context.

3. STATE-OF-THE-ART OVERVIEW

This section motivates the importance of EAGLE. EAGLE calls for uncertainty-aware and partial models. Uncertainty here corresponds to a metric system for measuring the incompleteness of the models that is due to the technique used to elicit the models from real artifacts (e.g., code or running system) and their context. EAGLE systems opportunistically integrate pieces of software as available in a non-ideal world: this leads to accept incomplete information, hence accepting systems that represent the strictly necessary solution for satisfying the specified goal, possibly also in face of risks identification and prioritization. Thus, goal-oriented validation is another key aspect for EAGLE. As discussed below, there exist many methods and techniques to account for uncertainty while developing software systems. All of them operate within different domains and consider uncertainty at different abstraction levels by also exploiting different software models. In this direction, one of the aims of EAGLE is to combine/extend existing techniques and methods into a unified uncertainty-aware framework. Therefore, our state-of-the art overview is organized in several parts: Section 3.1 presents approaches addressing the problem of deriving partial models from implemented systems. Section 3.2 the models@runtime approach and Section 3.3 presents automatic connector synthesis to support software integration and coordination. Finally, Section 3.4 presents functional and non-functional verification and validation under uncertainty.

3.1 Derivation of partial models

Several approaches have recently addressed the problem of deriving partial behavioral models from implemented systems. In [7], we propose a method that combines synthesis and testing techniques in order to automatically derive the behaviour protocol of a web-service out of its WSDL interface. In [34], the authors propose a novel synthesis technique that constructs partial behavioural models in the form of Model Transition Systems from a combination of safety properties and scenarios. In [26], the authors describe a technique to automatically generate behavioral models from (object-oriented) system execution traces. The work described in [21] aims to infer a formal specification of stateful black-box components that behave as data abstractions by observing their run-time behavior. The approach described in [35] analyzes Java code to elicit object usage patterns. The authors of [4] present an approach for inferring state machines with an infinite state space. The author of [27] use constraint solving to carry on learning-based black-box testing. In [14] the authors propose tools and techniques to automatically derive models from running open source software systems in order to enable the simulation of their upgrades and to detect possible configuration inconsistencies.

3.2 Models@runtime

The models@runtime approach [8] seeks to extend the applicability of models produced in Model Driven Development (MDD) approaches to the run-time environment. An example of design models application at run-time has been proposed within the PLASTIC project, where we have been involved in. The PLASTIC development process [2] relies on model-based solutions to build adaptable context-aware service-oriented applications. Opportunistic reuse of heterogeneous pieces of software, context awareness, run-time evolution, adaptiveness and uncertainty represent challenges that can be addressed by adopting a models@runtime approach [8]. Modeling techniques coupled with MDD capabilities such as model transformation and code generation provide viable means to enable system monitoring, model analysis and adaptation at run-time [23].

3.3 Automatic connector synthesis to support software integration and coordination

The first approaches to connector synthesis appeared in the 90s in the control theory domain [31] and, thereafter, they have been revised to fit the domain of software (embedded) systems [1, 3]. The aim of these approaches is to automatically synthesize a controller that restricts the system behavior so as to satisfy a given specification. In [11, 30], LTSs are used to model the I/O behaviour of components and automatically synthesize a set of constraints on the components' environment that allow deadlock avoidance. In [33], we show how to automatically derive either a centralized or distributed connector from a specification of the components' interaction and of the requirements that the composed system must fulfill. However, these approaches do not take into account both possible run-time changes in the environment and non-functional requirements of the system to be integrated. EAGLE aims at tackling the problem of automatically synthesizing integrators at run-time under uncertainty.

3.4 Functional and non-functional verification and validation under uncertainty

The idea of moving V&V activities at run-time [5] has been often realized by introducing monitoring activities both for functional and non-functional properties, and more recently by moving testing to on-line [6]. Uncertainty in EAGLE calls for compositional

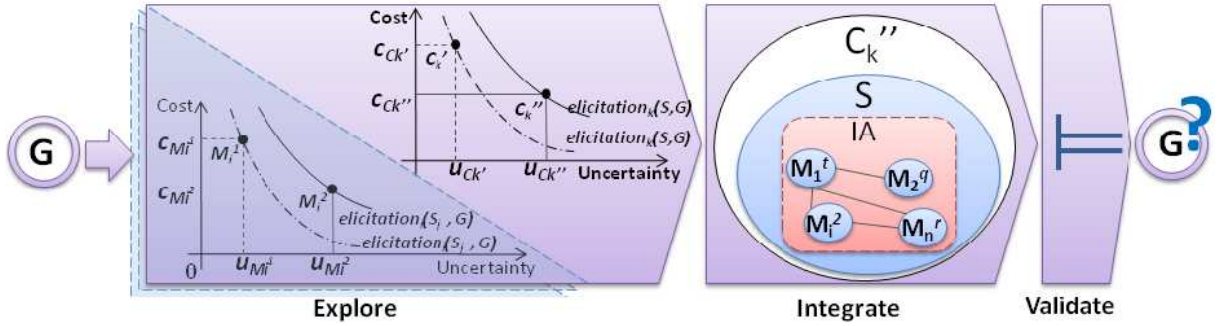

Figure 1: Explore, Integrate, and Validate phases

verification and validation (V&V) techniques that permit to perform partial V&V (based on the information currently available) and to instrument the system so to be able to support on-line V&V. Many works have been proposed in compositional verification and in particular in assume-guarantee reasoning, such as [13, 22] and to [15]. Bayesian models (such as Bayesian Networks [29]) can be considered as the stochastic counterpart of the assume-guarantee paradigm. In this direction, an example of bayesian approach for modeling the reliability of a software component-based system, given the reliability of its components, has been presented in [32]. More sophisticated stochastic models can be used to take into account uncertainty in non-functional validation processes. Hidden Markov Models (HMM) [16] are typically used to model systems that have markovian characteristics in their behavior, but that also have some states (and transitions) for which only limited knowledge is available. Finally, theories [24] and techniques [9] for compositional approaches to testing have been investigated.

4. METHODOLOGY

In the EAGLE approach, opportunism and goal-orientation imply that software production process qualities change from more traditional aspects of efficiency and resource optimization, to adaptability and fit-for-purpose, even accepting inefficient usage of resources and redundancy. The uncertainty degree of a piece of software, with respect to a given goal G, can be determined by using explorative techniques that range from standard analysis and validation techniques to machine learning techniques [17]. Let S_1,...,S_k be pieces of software that can be used to build a system S that satisfies G. In general different subsets of $\{S_1,...,S_k\}$ will suffice to satisfy G. The result of an explorative phase is a set of approximation models $M=\{M_1,...,M_n\}$ of different types, i.e., a set of observations of S_1,...,S_k.

Each type of model M_i is elicited by adopting an explorative technique, e.g., $elicitation_i(S_j, G)$, and represents an observation of an S_j. Each explorative technique can obviously be instantiated in different ways, depending on the desired semantics of the target model. As an example, an explorative technique can elicit automata from Java applications, but the specific instantiation of the technique will determine the semantics of the elicited automata. Hence, we denote by M_i^1,...,M_i^q the set of models that can be elicited with the explorative technique i. Each model shall have associated its own metric system for measuring the degree of uncertainty $u_{M_i^j}$. Thus, a piece of software and a goal can have associated different models, i.e., different observations, each with its own degree of uncertainty. Moreover, each elicited model M_i^j has a cost $c_{M_i^j}$ that represents a quantitative measure of the effort to elicit M_i^j with an uncertainty degree $u_{M_i^j}$. In the Explore box of Figure 1 we represent this scenario, where different curves for the same explorative technique represent different instances of the technique, each able to elicit models with different costs and uncertainty degree (along the curve). This box has a certain multiplicity (as represented by the dashed box boundaries) given by the multiplicity of explorative techniques and their instances. Similarly, a set of types of models $C=\{C_1,...,C_n\}$ can be elicited for the context and analogous definitions of uncertainty and cost metrics can be introduced for it.

The *integration* step shall support the producer in creating the most effective (to the goal G) integration means that takes into account the uncertainty degree, and the associated cost, of each single elicited model. This means that during the integration step, by reasoning on their elicited models and further accounting for the trade off between uncertainty degree and cost, the candidate pieces of software (i.e., a subset of $\{S_1,...,S_k\}$) need to be selected. Thus, an integration architecture *IA* shall be provided, possibly automatically. *IA* plays a crucial role in influencing the overall uncertainty degree of the final integrated system S, as different *IAs* may result in different uncertainty degrees for S. These observations lead to a definition of the integrated system uncertainty u_S as a function of the software models that have been selected in the explorative step, suitably integrated according to the *IA* and the context models.

A validation step shall assess the quality of the integrated system with respect to the overall system goal G and the current elicited context models. If the final assessment is not satisfying then the framework should permit to iterate the process either to select different pieces of software, or to reduce the uncertainty degree of some (already in place) models, or to modify the overall *IA*.

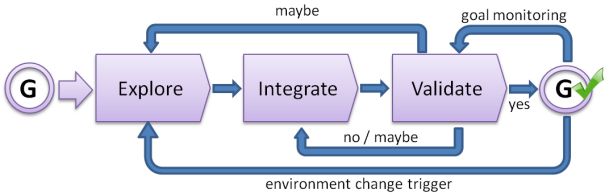

Figure 2: Explore, Integrate, and Validate cycle

Figure 2 shows the perpetual activity flow of the foreseen process. In particular, if validation says that the goal is not achieved (**no**) then a new integration architecture is considered. The aim is to act on the integration means, e.g., connectors, so to avoid interactions that prevent the achievement of the goal. If it cannot be assessed whether the goal is achieved or not (**maybe**) either a new integration architecture is considered, or more accurate software models are incrementally elicited. For instance, under cost and resource constraints, the validation may not be able to reach a feasible result. In this case, a new integration architecture can be designed to make the validation feasible. Furthermore, a lack of information in the considered models may lead to a meaningless validation, thus claiming for more accurate models to be elicited. Finally, if the validation step shows that the goal is achieved (**yes**) then an entire explore-integrate-validate iteration is terminated. Whenever changes in the monitored environment occur, the iteration of a new cycle is triggered.

5. CONCLUSION

Overall EAGLE is very ambitious. Each of the above described objectives launches key challenges that will characterize and steer the research in software in the next future. Each objective cannot be addressed in isolation from the others but needs to be cast in the scope of a holistic explore-integrate-validate process life cycle. The software process we envisage lifts uncertainty to the level of a first class notion among the others and asks for explicit characterizations of the software artifact properties and assumptions that take into account their incompleteness, as described above. EAGLE aims at defining and developing novel theories, models, model-driven techniques, and tools for application production, verification and validation, and integration code synthesis, which are all characterized by the ability of tackling the unknown. From the forum, we expect to exploit possible feedbacks from both computer scientists and practitioners to evaluate the exploitation, in the near future, of the project results with respect to a multitude of contexts both research-wise and industrial-wise.

6. REFERENCES

[1] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, 1995.

[2] M. Autili, L. Berardinelli, V. Cortellessa, A. Marco, D. Ruscio, P. Inverardi, and M. Tivoli. A development process for self-adapting service oriented applications. In *Proc. of ICSOC '07*, pages 442–448, 2007.

[3] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems (extended abstract). In *IFIP TCS 2004*, volume 155. 2004.

[4] T. Berg, B. Jonsson, and H. Raffelt. Regular inference for state machines using domains with equality tests. In *Proc. of FASE'08/ETAPS'08*, pages 317–331, 2008.

[5] A. Bertolino, G. Angelis, L. Frantzen, and A. Polini. Software engineering. chapter The PLASTIC Framework and Tools for Testing Service-Oriented Applications, pages 106–139. Springer-Verlag, Berlin, Heidelberg, 2009.

[6] A. Bertolino, G. De Angelis, and A. Polini. (role)CAST : A Framework for On-line Service Testing. In *Proc. of WEBIST'11*, 2011.

[7] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proc of ESEC/FSE '09*, 2009.

[8] G. Blair, N. Bencomo, and R. B. France. Models@run.time. *Computer*, 42:22–27, 2009.

[9] C. Blundell, D. Giannakopoulou, and C. S. Păsăreanu. Assume-guarantee testing. *Softw. Eng. Notes*, 31, 2005.

[10] B. W. Boehm. Software risk management: Principles and practices. *IEEE Softw.*, 8:32–41, January 1991.

[11] C. Canal, P. Poizat, and G. Salaün. Synchronizing behavioural mismatch in software composition. In *FMOODS*, pages 63–77, 2006.

[12] S. Ceri, D. Braga, F. Corcoglioniti, M. Grossniklaus, and S. Vadacca. Search computing challenges and directions. In *Proc of ICOODB'10*, pages 1–5, 2010.

[13] J. M. Cobleigh, D. Giannakopoulou, and C. Pasareanu. Learning Assumptions for Compositional Verification. In *Proc. of TACAS 2003*, number 2619 in LNCS, 2003.

[14] R. Di Cosmo, D. Di Ruscio, P. Pelliccione, A. Pierantonio, and S. Zacchiroli. Supporting Software Evolution in Component-Based FOSS Systems. *Science of Computer Programming*, 76(12), 2011.

[15] J. Dingel. Computer-Assisted Assume/Guarantee Reasoning with VeriSoft. In *Proc. of ICSE2003*.

[16] Y. Ephraim and N. Merhav. Hidden markov processes. *IEEE Transactions on Information Theory*, 48:1518–1569.

[17] M. D. Ernst and J. H. Perkins. Learning from executions: Dynamic analysis for software engineering and program understanding, Tutorial at ASE 2005.

[18] J. R. Galbraith. *Designing Complex Organizations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1973.

[19] D. Garlan. Software engineering in an uncertain world. In *Proc. of FSE/SDP'10*, pages 125–128, 2010.

[20] C. Ghezzi. ERC Advanced Investigator Grant N. 227977 [2008-2013].

[21] C. Ghezzi, A. Mocci, and M. Monga. Synthesizing intensional behavior models by graph transformation. In *Proc. of ICSE '09*, pages 430–440, 2009.

[22] D. Giannakopoulou, C. S. Pasareanu, and H. Barringer. Component verification with automatically generated assumptions. In *ASE journal, 12(3): 297-320*, 2005.

[23] H. J. Goldsby and B. H. Cheng. Automatically generating behavioral models of adaptive systems to address uncertainty. In *Proc. of MoDELS '08*, pages 568–583, 2008.

[24] D. Hamlet. *Composing Software Components: A Software-testing Perspective*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[25] P. Inverardi. Software of the future is the future of software? In *Proc. of TGC'06*, pages 69–85, 2006.

[26] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *Proc. of ICSE '08*, pages 501–510, 2008.

[27] K. Meinke. Automated black-box testing of functional correctness using function approximation. In *Proc. of ISSTA '04*, pages 143–153, 2004.

[28] J. Mula, R. Poler, J. Garcia-Sabater, and F. Lario. Models for production planning under uncertainty: A review. *IJPE*, 103(1):271–285, 2006.

[29] M. Neil, N. Fenton, and M. Tailor. Using bayesian networks to model expected and unexpected operational losses. *Risk Analysis*, 25(4):963–972, 2005.

[30] R. Passerone, L. de Alfaro, T. A. Henzinger, and A. L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: two faces of the same coin. In *Proc. of ICCAD '02*, pages 132–139, 2002.

[31] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81 –98, jan 1989.

[32] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj. A bayesian approach to reliability prediction and assessment of component based systems. In *Proc. of ISSRE '01*, 2001.

[33] M. Tivoli and P. Inverardi. Failure-free coordinators synthesis for component-based architectures. *Sci. Comput. Program.*, 71(3):181–212, May 2008.

[34] S. Uchitel, G. Brunet, and M. Chechik. Synthesis of partial behavior models from properties and scenarios. *IEEE Trans. Softw. Eng.*, 35:384–406, May 2009.

[35] A. Wasylkowski, A. Zeller, and C. Lindig. Detecting object usage anomalies. In *Proc. of ESEC-FSE '07*, 2007.

[36] R. W. White and R. A. Roth. *Exploratory Search: Beyond the Query-Response Paradigm*. Synthesis Lect. on ICRS. Morgan & Claypool Publishers, 2009.