UML profile for modeling product observation.

UML Profile for Modeling Product Observation

Mathias Funk, Piet van der Putten, Henk Corporaal
Dept. of Electrical Engineering, Electronic Systems Group
Technical University Eindhoven, The Netherlands
Email: [m.funk, p.h.a.v.d.putten, h.corporaal]@tue.nl

Abstract

Nowadays interactive electronics products offer a huge functionality to prospective customers, but often it is too huge and complex to be grasped and used successfully. In this case, customers obviate the struggle and return the products to the shop. Also the variability in scope and features of a product is so large that an up-front specification becomes hard if not impossible. To avoid the problem of an inadequate match between customer expectations and designer assumptions, new sources of product usage information have to be developed. One possibility is to integrate observation functionality into the products, continuously involving real users in the product development process. The integration of such functionality is an often overlooked challenge that should be tackled with an appropriate engineering methodology. This paper presents on-going work about a novel design for observation *approach that supports early observation integrations and enables the cooperation with various information stakeholders. We show how observation can be embedded seamlessly in a model-driven development process using UML. An industrial case-study shows the feasibility of the approach.*

1. Introduction

Complex innovative electronic products often fail to satisfy customers' needs. Products are too complicated, thus too cumbersome to use. The inherent functionality is often not relevant to user needs and expectations. Increasing numbers of returned technically sound products support this [1]. On the other hand, nowadays products are hard to specify because of their high complexity and because of rapidly changing user demands. Also, due to faster cycles, the product creation process cannot benefit from traditional feedback channels any more. While a couple of years ago, a product technology could reach maturity within 10 to 20 cycles, thus allowing for gradual improvements, todays products have to accomplish the same within three cycles. Obviously, delivering a mature product in this setting becomes difficult.

Accordingly, complex interactive products should be built for rapid changes. Products can be adapted to changing needs during development and even after release, in terms of firmware updates and and the like. Still, targeting the product for a certain user base is a major problem in the industry [1]. One reason for this is the lack of usage information which is reliable enough to base the further development of the product on.

Our approach towards this problem is to build observation modules into products. These products are given to selected key testers who use the products in their habitual environment - an approach promising to yield more representative data than usability labs. The built-in observation modules can be configured remotely and observe parts of the system including user interaction, system performance and potentially user satisfaction with the system provided functionality.

This paper concentrates on the integration of such observation facilities into products. We propose a model-driven technique to do this in an efficient and structured way which is tailored to current system development practices. After pointing at related work, we introduce product observation together with an industrial case-study. Subsequently, we show how observation-related parts of the system can be modeled by means of a novel UML profile. An overview on the development approach for observation integration shows the application of the modeling. The flow from system models to the final runtime which features a built-in observation system is described. The paper ends with a conclusion and an outline of future steps.

An extended version of this paper covers the UML profile for observation in full length [2].

2. Related work

The remote monitoring of products has been done before, ranging in scope from the monitoring of cars to building automation, computer programs, mobile devices, and websites [3], [4], [5], [6]. However, our research is different in two important aspects: First, in our approach we assume that information stakeholders are not willing to use complex programming paradigms to achieve the sought-after data, therefore we use a visual language to specify observation behavior in a domain-specific way. Second, the integration of observation functionality into the target system is described in a software engineering process which is, in our opinion, necessary for widespread use. On the technical level we rely on the proven model-driven engineering approach, but also try to apply more agile modeling techniques like model interpretation [7] that allows for dynamic adaptation of runtime systems without the need for client compilation support. The modeling of observation systems is performed using the Unified Modeling Language (UML) [8] and, more precisely, its profile extension mechanism [9].

Our approach towards "design for observation" is related to "design for test". The goal is to enable evaluation of systems as early as possible and throughout the development process. Significant effort has to be spent before valuable data emerges. However, design for test targets mainly the scope of specification correctness within the system, whereas design for observation aims at the reduction of failures during interaction between user and system.

3. Product usage observation

Our approach separates the concerns of (i) product or system development and (ii) the specification of what to observe and how to present the collected data. In its application, product observation involves accordingly two roles: the first role is a system developer, concerned with the integration of the observation module into the product. The second role, the information stakeholder, specifies observation in an easy and straight-forward process. For information stakeholders, the proposed approach opens a dedicated information channel which provides potentially high quality data. Even

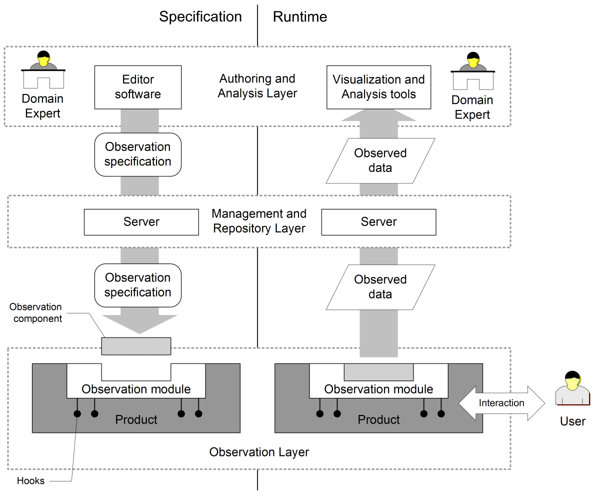

Figure 1. Technical observation system overview

more importantly, the observation behavior can be adapted remotely to changing information needs.

An observation system [10] consists of three main layers: the *authoring and analysis layer (AAL)*, the *management and repository layer (MRL)*, and the *observation layer (OL)* (cf. Figure 1). On the first layer, it can be specified what to observe and how to process and present the collected information. The MRL plays mainly the role of a middleware between specification of observation and observation itself, it transports observation specifications towards the OL and the observed data from products back to the AAL. The OL is the place where the actual observation is carried out within the product instances. From the development perspective, two interesting things happen here: First, *hooks* have to be integrated into the product. They represent places that can be observed, that is, they are proxies to the actual places in hardware and software where the actual data is generated. This encapsulation helps to maintain a consistent interface from product to observation module. Second, observation specifications coming from the AAL are transformed into executable runtime structures and represent the logic of observation in a certain scenario. The latter aspect is beyond the scope of this paper and has been covered in [11].

3.1. A priori case study

We carried out case-studies to explore the domain of observation and build experience for developing an architecture. One of the studies together with a large Dutch consumer electronics manufacturer shall be described briefly as an introductory example to the domain of product usage observation.

Subject of the observation was a working prototype of an "internet on television" product with a couple of novel features. Regarding those features, the company had no market experience, so the main goal was to explore the relevance of such a product to customers. We integrated the observation module partly by reverse-engineering the system for appropriate observable items, partly by intrumenting available source code to provide data about usage. Eventually, 20 product instances were given to key testers located in 8 countries world-wide. These people were asked to embed the product into their home lifestyle and use it regularly for 6 weeks. During that time, the observation system collected in total 800.000 data items and moreover, we were able to test the use case of changing observation requirements successfully. This and other experiments proved the applicability of the approach and motivated the development of an engineering approach for observation integration.

4. Modeling observation

Modeling of observation systems can be done in the Unified Modeling Language (UML). This language is an industry-wide standard for modeling of hardware and software systems. UML models are widely understood by developers in the community, and the modeling process benefits from extensive tool support. UML offers a light-weight extension mechanism, *profiles*, that is suitable for building domain-specific UML models. This means to project domain language semantics onto UML by technically extending it with a dedicated set of new concepts.

The observation profile as shown in Figure 2 is basically divided into five sets of subprofiles that correspond to the layers of an observation system (cf. Section 3). While the upper 4 profile packages are entirely concerned with the *observer* side of the data collection process, the *observation execution* profile package at the bottom of the figure involves both *observer* and *observee* roles. There are two sets of stereotypes, each concerning one of the two roles in the *observation integration* sub profile. This is especially interesting as the thin line between the sets can be drawn right through this profile. Both sets have a close relationship as observed information is transmitted directly between system parts which have complementing stereotypes

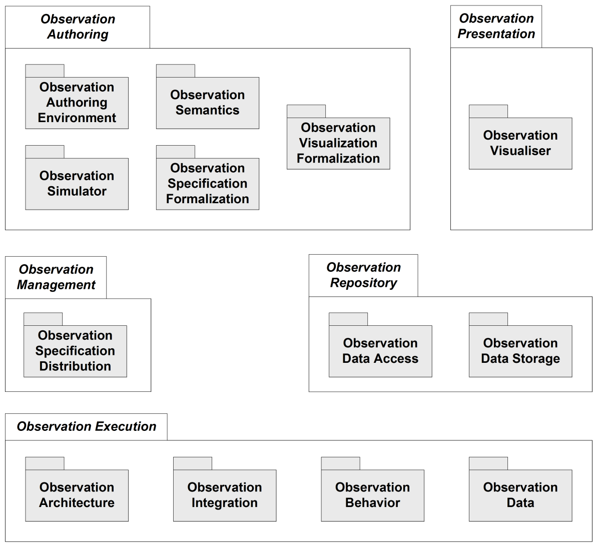

Figure 2. Observation profile (package view, dependencies omitted)

applied to them. Inside the *observation execution* package there are two packages concerned with the *architecture* and *integration* of observation into products. Another package deals with the observation *behavior* at runtime, and the remaining package provides structures for the observation *data*.

4.1. Observation integration subprofile

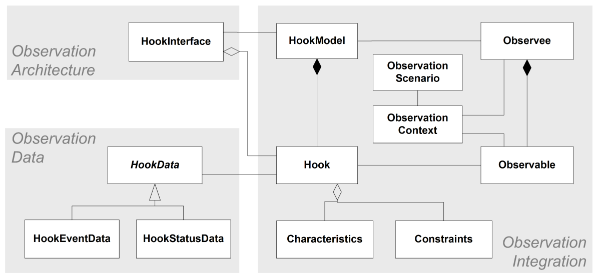

Figure 3. Integration profile

In the specification of observation, *hooks* are used as information sources. However, hooks are only abstract places where information can be perceived. Therefore, on the system level, a «Hook» is realized as a proxy element that encapsulates the combination of an «Observable» element and its observation-related properties, such as «Characteristic» and «Constraint». Characteristics cover timing properties and data types, constraints describe runtime limitations of the observable. This meta-information can be used to implement predictable observation modules, or to simulate observation behavior prior to deployment.

Hooks and observables are basically two different views on the same entity. From the *hook* side, only observation-related properties are shown and other information, e.g. about the implementation, is hidden - vice versa from the *observable* side. Both concepts are aggregated in respective concepts, «HookModel» and «Observee». While the *Hook-model* denotes a collection of hooks, the *Observee* is a system part which contains «Observable» elements, but is itself not directly observable. This stereotype can be used early in the development to annotate system parts that should be observed, and can be refined later to actual «Observable»s. Another stereotype of the integration profile is the «ObservationContext» which represents contextual information belonging to observable or observee. This information can (i) determine how the observable behaves, generates data, and responds to triggering, and (ii) it can be part of the raw observation data that is generated by the observable. All context information depends on the «ObservationScenario». Such a scenario is a usage setting and contains information about the enviroment the product is used in as well as the user who interacts with the product.

To further explain the relationship between hooks and observables, interaction patterns in the observation domain are shown in Figure 4. The nature of hooks, being either *self-triggering*, *externally triggered*, or both, suggests basically two interaction patterns. The *self triggering* and the *externally triggered* patterns are explained by using the aforementioned stereotypes of «Observable» and «Hook».

The first pattern is suitable for hooks which are *self-triggered*, that is, the observable system structure autonomously triggers the respective hook object whenever new information is perceived and should be fed into the observation system. In this pattern the responsibility of taking action lies on the observee's side.

The second pattern deals with hooks that have to be *triggered externally* to produce data. Here, the hook object has a link to the observable structure, e.g. in the form of a public operation, and can trigger the observable. In the rare case that an observable has both characteristics, a combination of those patterns is also possible.

Hooks, as their proxy nature suggests, connect

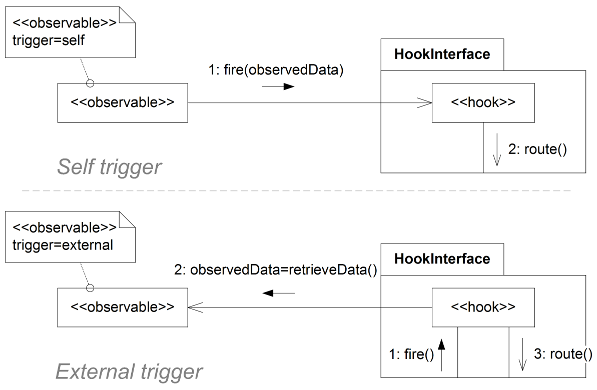

Figure 4. Interaction patterns, note the communication directions

observables to the respective interface in the observation module, the «HookInterface» as shown in Figure 3. The observable element delivers raw data to the interface, and inside the module this interface presents the data to the observation component. The component subsequently processes the raw data according to the specified observation *behavior*. Obviously, the resulting data is determined to a large extent by the observation system input coming from hooks, thus the strong connection to the «HookData» stereotype (cf. Figure 3).

4.2. Observation architecture subprofile

The «HookInterface» is one of the main parts of the *observation architecture* profile shown in Figure 5. The interface stereotype is a part of the «ObservationModule», namely being a sub stereotype of «ObservationSubModule». Other sub modules are concerned with the communication between observation and repository layer (cf. Figure 1).

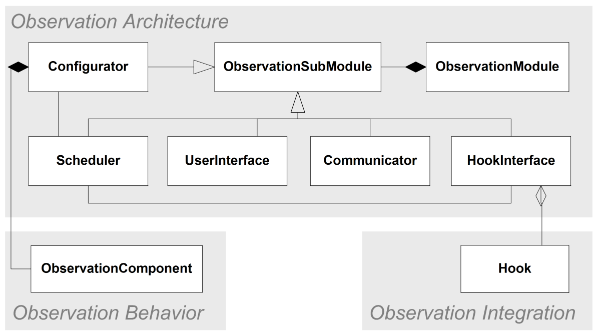

Figure 5. Architecture profile

Two submodules deal with runtime behavior of specified observation, the observation component. The «Configurator» receives an observation specification and translates it into an executable «ObservationComponent» which is run by the «Scheduler». The latter module is responsible for triggering of hooks and the synchronization of concurrent events.

Both the architecture and the integration profiles are shown here in an overview. Especially the architecture profile contains more elements, that structure the big building blocks depicted in Figure 5. In the next section, the usage of the observation profile will be described in an example development flow.

5. Development flow

The development of an observation system is a model-driven process which is mainly supported by the observation profile as shown above. The actual usage of the profile shall be explained in the context of development. Figure 6 shows the basic development flow for system parts in the observation layer following the model-driven architecture [12] approach: a platform-independent model is transformed into a platform-specific model, both extended by observation-specific meta information. Then an additional weaving step integrates observation facilities, also modeled in UML, into the platform-specific system model. Although these observation facilities could be modeled directly in the system model, for later reuse, the modeling of observation in a separate model is advisable. The result of this step is a system model enriched with observation facilities. This new model can directly be used in the subsequent code generation step.

Additionally, a *hook model* is obtained from the enriched system model. A hook model describes the nature of all hooks present in the system which can be used in the specification of observation. It is necessary for (i) documentation of the observation capabilities of the system and (ii) for the later specification phase where it defines which hooks can potentially be used and which properties they might have.

The flow depicted in Figure 6 starts with the application of the observation profile to system models. Both platform-independent and platform-specific UML models can make use of this profile by application of the respective stereotypes. It is advisable to integrate observation as early as possible into the development, going from rather coarse-grained stereotypes like «Observee» to more fine-grained ones such as «Observable». Generally it

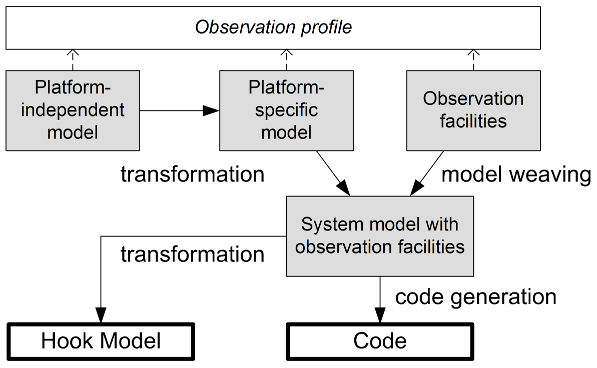

Figure 6. Observation integration modeling flow

should be possible to attach stereotypes to parts of both models as not all observation-related modeling is necessarily platform-specific, and vice versa.

In order to weave system model and observation model, first the observation facilities are retrieved from the observation model and are inserted into the system model. This is a simple merging of model elements on the package level. In a second step, all «Observable» stereotypes on parts of the extended system model are identified and checked against the observation facilities. This second step of the weaving algorithm works as a tree traversal, checking all model elements for applied observation stereotypes. In case an element has the stereotype «Observable» applied to it, it depends (i) on the nature of the element and (ii) on the interaction pattern (see section 4.1), what kind of connection is introduced to the observation module (also see below). In any case, a corresponding «Hook» is introduced that encapsulates the different interaction behaviors, data types and timing of observables.

The code generation step first creates structural code for all model elements and large parts of the observation module can already be generated. This applies especially, if an implementation of the observation module exists already for the target platform. Moreover, a subset of the hooks might be realized in a platform-independent fashion, and thus can be reused. Otherwise, at least glue code for hook implementations can be generated automatically from the extended system model. In this case the code generation step simply continues the work of the weaving algorithm on the code level: implementations are created for «Observable» and «Hook» as far as possible. Finally, the developer has to review the glue code in the observables and add a few instructions to make the observables deliver actual data.

6. Conclusion & Future work

Companies experience a lack of reliable, product-specific usage information. We address the information deficit by building observation modules into products that are capable of providing usage information directly from products in the field. A case-study shows the applicability of the approach. Observation integration will potentially have a strong impact on the development of future innovative products, thus the need an engineering methodology. This paper introduces a model-driven technique to integrate observation functionality into products by means of a novel observation profile. System models are enriched with observation-specific concepts and a dedicated model of supportive observation facilities is weaved into the system model using the concepts as semantic links. The result is an extended system model that can be used in a final code generation step and that documents the observation capabilities of the system. The technique introduced here simplifies the development tasks necessary for observation integration, thus reducing the effort for integration. It helps to automate the process of observation specification and data collection. Furthermore, we see observation integration not as a simple parallel development task only, but as a potential driving force behind a new development paradigm: *design for observation*. This involves observation as a first class development aspect and helps to provide a solid basis for extensive, but meaningful product information presented to information stakeholders in a comprehensive way.

Acknowledgments

This work is being carried out as part of the "Managing Soft-Reliability in Strongly Innovative Product Creation Processes" project, sponsored by the Dutch Ministry of Economic Affairs under the IOP-IPCR program.

References

[1] E. den Ouden, L. Yuan, P. J. M. Sonnemans, and A. C. Brombacher, "Quality and reliability problems from a consumer's perspective: an increasing problem overlooked by businesses?" *Quality and Reliability Engineering International*, vol. 22, no. 7, pp. 821–838, 2006.

[2] M. Funk, P. H. A. van der Putten, and H. Corporaal, "UML profile for modeling system observation," Eindhoven University of Technology, Tech. Rep. ESR-2008-09, 2008. [Online]. Available: http://www.es.ele.tue.nl/esreports/

[3] H. Hartson and J. Castillo, "Remote evaluation for post-deployment usability improvement," *Proceedings of the working conference on Advanced visual interfaces*, pp. 22–29, 1998.

[4] D. M. Hilbert and D. F. Redmiles, "An approach to large-scale collection of application usage data over the internet," *icse*, vol. 00, p. 136, 1998.

[5] K. Kabitzsch and V. Vasyutynskyy, "Architecture and data model for monitoring of distributed automation systems," in *1st IFAC Symposium on Telematics Applications In Automation and Robotics*, Helsinki, 2004.

[6] E. Shifroni and B. Shanon, "Interactive user modeling: An integrative explicit-implicit approach," *User Modeling and User-Adapted Interaction*, vol. 2, no. 4, pp. 331–365, Dec. 1992. [Online]. Available: http://dx.doi.org/10.1007/BF01101109

[7] J. Estublier and G. Vega, "Reuse and variability in large software applications," in *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering.* New York, NY, USA: ACM, 2005, pp. 316–325.

[8] Object Management Group, "Unified modeling language," 2006. [Online]. Available: http://www.uml.org

[9] D. D'Souza, A. Sane, and A. Birchenough, "First-class extensibility for UML-profiles, stereotypes, patterns," in *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, R. France and B. Rumpe, Eds., vol. 1723. Springer, 1999, pp. 265–277. [Online]. Available: citeseer.ist.psu.edu/dsouza99firstclass.html

[10] M. Funk, P. H. A. van der Putten, and H. Corporaal, "Specification for user modeling with self-observing systems," in *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, Saint Luce, Martinique, 2008.

[11] M. Funk, P. H. A. van der Putten, and H. Corporaal, "Model interpretation for executable observation specifications," in *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering.* Knowledge Systems Institute, 2008.

[12] D. Frankel, *Model-Driven Architecture.* New York, NY, USA: OMG Press / Wiley, 2003.