Zusammenfassung/Abstract

Business process modeling solutions aim at tightly involving the application expert in the software development process to avoid common and costly 'communication accidents' during requirements engineering and to decrease time to market. Their popularity, in particular of the recent standard BPMN 2.0, clearly indicates the need for this new involvement. At the same time there are unexpected hurdles when it comes to dealing with integration, (runtime) variability, and interoperability. *Higher-order process engineering* (HOPE) elegantly overcomes these hurdles as even in its simplicity-oriented version it allows for a powerful plug&play fashion, where processes and services can be moved around just like data. This is reminiscent to standardization efforts known from hardware like the *universal serial bus* (USB).

Computer-supported processes increasingly influences our daily life: be it for shopping, travel planning, travelling itself, tax declaration, or as part of our business life, we are confronted all the time with different computer/web applications of enormously varying quality and flexibility. Indeed, there are very different solutions for almost the same problem, even by the same provider, depending on the particular profile of a particular user/customer situation. The variance concerns not just the look-and-feel but also the required user process, and, at the technical level the application programming interfaces (APIs) for the interoperation between systems. As a consequence, users continuously have to fight with this historically grown but unintended technical diversity, and producers with exploding maintenance costs.

Increased interest in business process modeling came with BPMN 2.0, due to its promise to make application-level business processes directly executable. It looked like an almost universal cure, as it seemed to close the semantic gap, the main source of misunderstandings between business and IT. Looking closer, this dream becomes true only for very specific scenarios. In particular, it still does not support variability, as it lacks means to manage process variation and variant-rich systems. The need for variant-rich systems is often addressed via static approaches in general subsumed by the notion of *product-lines* in the literature. All current BPM standards like BPEL, BPMN and even the recent BPMN 2.0 are constrained to fixed bindings between business activities in a model and the services or substructures they refer to. Thus each variant has to be modeled explicitly via decision points that are modeling pendants to *if*- and *switch*-clauses, in comprehensive but very large models, or maintained individually, as a collection of separate entities. Both alternatives cause severe problems: whereas the comprehensive modeling leads to unmanageably large models, the maintenance and support of many structurally similar individual variant models is a nightmare. This is far away from the ideal of simplicity that made business process modeling so appealing.

Enhancing BPM with a *simplicity-oriented* version of *higher-order process passing* breaks the spell. Being higher-order, it supports a very flexible form of (type-based) process integration. It also introduces a new discipline of variability modeling that captures variants comprising functionality that was still unknown at the process' start. Already

deployed and even running processes can be seamlessly enhanced with new functionality, without touching their code base.

In fact, based on the higher-order concepts, (new) services and (component) processes can be selected, modified, constructed and then safely passed *as if they were data*. Though unlike data they may be plugged into activities and executed (played) dynamically. This *plug&play* approach allows one to add new services, components, and processes without the need to change the system or interrupt the running processes. Even if this flexibility is not essential and all functionalities are known in advance, it leads to drastic size reductions and better understandability of the models.

Controlling variability means controlling the exploding wealth of combinations, a concern orthogonal to computational aspects. Thus in particular at the level of (business) process modeling, variability concerns and computational aspects should be separated in favor of simplicity and understandability.

Thus, simplicity-orientation in this regard refers to the user-centeredness – focused on application experts – of the approach hiding the complexity going hand in hand with achieving compatibility between components essential for plug&play semantics. In this regard two different types of abstraction layers support a strict enforcement of *separation of concerns*: *hierarchical modeling* and *preconfiguration of activities*.

The need for hierarchical modeling is owed to the fact that on the one hand systems are getting more and more complex, resulting in the need to break the specifications down into manageable pieces in a *divide and conquer* fashion. On the other hand, with the increasing globalization of processes and systems, e.g. end-to-end-processes that naturally comprise various management levels, the number of participants involved in the development process is also steadily increasing. This results in a complex network structure for the participants, where the technical expert on one level is at the same time the application expert on the next lower level – in terms of getting more technical – and so forth. Preconfiguration of activities further helps to guard the modeler from the complexity of data-flow handling as well as from doing repetitive work.

Compatibility is achieved via sophisticated (type-aware) interface descriptions of activities leading to compositionality. Data-dependencies are consequently modeled via a common execution context sharing resources between activities. The models are hierarchical starting on a domain-specific level, leading down to a technical level. The process components on the technical level base on structures of a conventional programming language (i.e. the *target language*). The idea is to give more power to application experts as well as avoiding *technological breaks*, leading to more complete process models which

- are a better basis for communication with technical experts,
- allow for more concise validations on the modeling level, and
- need no round-trip engineering as they are based on semantically well-defined structures of a target language and are therefore directly executable.