Emergence as Competitive Advantage
Engineering Tomorrow's Enterprise Software Systems

S. Frischbier[1], M. Gesmann[2], D. Mayer[2], A. Roth[3] and C. Webel[4]
[1]*Databases and Distributed Systems Group (DVS), TU Darmstadt, Darmstadt, Germany*
[2]*Software AG, Darmstadt / Saarbrücken, Germany*
[3] *SAP AG, Darmstadt, Germany*
[4] *Fraunhofer IESE, Kaiserslautern, Germany*

Keywords: Emergence, Business Process Modeling, Emergent Enterprise Software Systems, Interoperability, Adaptability, System-of-Systems, Software Evolution, Event-based Systems, Service-oriented Architectures, SOA, EBS, EDA.

Abstract: Companies rely heavily on complex software systems and tightly integrated supply-chains to serve their customers in increasingly fast changing markets. To gain competitive advantage in such a setting, companies must adapt their processes, products and inter-organizational relationships quickly to changing environments. In the future, enterprise software systems must be explicitly designed for flexibly switching intensive inter-organizational relationships and for rapidly implementing changes in requirements or context while retaining existing functionality and user acceptance. In this position paper we introduce the notion of emergence in enterprise software systems as a guiding principle. Emergent Enterprise Software Systems (EESS) will be capable of reacting to changes in the environment by adapting their behavior and exposing new functionality. The consequent challenges we face when designing, building and operating EESS are discussed.

1 ENTERPRISE SOFTWARE BEYOND DESIGN-TIME ADAPTABILITY

Companies rely heavily on large-scale enterprise software systems tailored to support their specific business processes. To meet rapidly changing expectations of customers and partners, processes and business relationships have to be changed at an increasing speed (Buchmann et al., 2010). Enterprise software systems today, however, still lack the level of flexibility needed in both intra- and inter-organizational process adaptation. With regard to cross-organizational interoperability, they are often designed in a company-centric way with implicitly shared semantics and models (Blair et al., 2011; Freudenreich et al., 2012). When implementing changed requirements or allowing third parties to extend today's systems this is rarely achievable in business real-time. Once designed and implemented using a vast variety of modular but specialized building blocks, today's enterprise software systems remain rather static. Altering single subsystems in isolation may even have unintended implications for the system as a whole - often forcing a redesign of the system landscape. This dramatically limits a company's flexibility to react to changing market situations and gain competitive advantage.

We introduce the notion of *emergence* to evolve software beyond design-time adaptability. In nature, emergence refers to the development of order and/or new behavior in response to changes in the environment based on local perception. In emergent software we expect the system to exhibit new behavior in response to changes of the environment. *Emergent Enterprise Software Systems* (EESS) must combine existing software paradigms (e.g., service-orientation) with reactive behavior (e.g., complex event processing) and self-x behavior (i.e., self-awareness and self-organization) into stable and reliable software systems. Furthermore, they have to take into account constraints of the business domain.

Project EMERGENT is a joint research project between academia and industry aiming at engineering EESS. As part of the government funded research

Frischbier S., Gesmann M., Mayer D., Roth A. and Webel C. (2012).
Emergence as Competitive Advantage - Engineering Tomorrow's Enterprise Software Systems.
In *Proceedings of the 14th International Conference on Enterprise Information Systems*, pages 181-186
DOI: 10.5220/0003970501810186

cluster *Software Cluster – Innovation for the Digital Enterprise* it brings together vendors and users of enterprise software with academic researchers.

The contributions of this position paper are threefold: i) we illustrate the shortcomings of today's enterprise software with an example based on our industrial partners' experience in §2; ii) we sketch out our vision of EESS in §3 and; iii) we discuss key challenges regarding their design, development and operation in §4 to be met in future work in our project (§5). Due to the expertise of the partners involved, this paper focuses on aspects of EESS related to architecture, business process modeling (BPM) and governance.

2 THE NEED FOR EMERGENCE

We illustrate the problems enterprise software vendors and users face today using a motivating scenario from the area of logistics. It involves the roles of online retailers, global parcel service providers and local retailers. We show that: i) fostering inter-organizational cooperation in this setting is not feasible with today's enterprise software; ii) business models and software architectures have to be synchronized to support end-to-end processes; iii) this leads to challenging requirements for tomorrow's enterprise software systems.

Metropolitan areas are the centers of our modern society as an ever increasing proportion of the world's population lives in an urban region. Public administration and enterprises have to react to new challenges like rising traffic, awareness for energy consumption or increased needs for transparency. In such a setting information needs to be efficiently and securely shared between all stakeholders and to be ad-hoc combined in innovative ways. Processes and supporting IT services need to allow citizens or companies to consume them easily and to build new value-added services on top of existing ones.

Fostering inter-organizational cooperation. In our scenario we assume an Urban Management Platform (UMP). Among many services for citizens, local companies, online retailers and local administration, an UMP provides services for urban logistics. This is depicted in Figure 1: End customers (*citizen network*) in urban regions can rely on a large number of local retailers with delivery services (e.g., for short distance delivery of food or flowers). These local *urban logistics* providers act efficiently and customer-centrally by getting short-run orders and delivering immediately. Pure *inter-city logistics* service providers in turn are essential for online retailers as they are highly efficient in distributing goods over long distances.

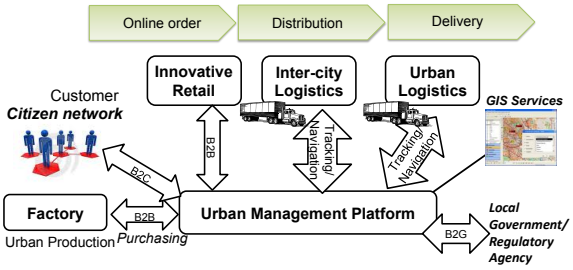

Figure 1: Urban Management Platform example.

However, they often fall short in delivering the expected user experience on the last mile (e.g., customers are not at home, delivery attempts fail several times and the customers have to pick up the parcel by themselves). Allowing for both types of service providers to dynamically set up or terminate business relationships based on the resources available would provide considerable benefits; especially end consumers would enjoy more convenient delivery conditions while keeping overhead for transportation low among suppliers. For example, online retailers could offer additional local goods (e.g., enhancing a birthday gift with flowers from a local store delivered in time) while the local retailer could offer the additional delivery of online ordered goods (e.g., delivering meals together with a movie on DVD ordered online).

Not feasible with today's enterprise software. Unfortunately, today's business software components and associated software engineering methods and tools do not allow for such flexibility. While large inter-city logistics service providers are supported by interoperable large-scale enterprise systems, the majority of small delivery service providers is often not supported by such systems. Therefore, the former have to rely on pre-defined local retailers with compatible systems to distribute their goods efficiently on short distance while the latter have to invest into systems which allow for the required interoperability. However, current systems are typically designed in an isolated way - they almost always require a lot of integration effort before efficient business relationships can be established and end-to-end processes can be supported.

We illustrate the challenge of supporting end-to-end processes in this setting by comparing the process steps of (fictitious) online and offline retail stores in Figure 2. On a high level of abstraction the main process steps seem to be similar: online order, distribution and delivery. However, when trying to establish business-relationships on-the-fly by hooking-up the software systems of the involved parties we are confronted with interoperability problems on the *pro-*

cess level (P1) and the *software level (P2)*: Processes and their underlying sub-processes differ extremely in detail depending on the company realizing each process step *(P1)*. As today's enterprise software is usually custom-tailored to the company-specific process implementations this leads to implicitly shared semantics and models being designed into the software. Trying to connect two modules (red squares in Figure 2) of different enterprise software systems results in interoperability problems on the software level which require a lot of manual integration effort *(P2)*. Both kinds of interoperability problems become explicit when we try to substitute the business process for the internet retailer (including delivery by selected intra-city providers via inter-city logistics providers) with the business process for the local retailers (local retailers directly hooking-up with small urban logistics providers for instant delivery).

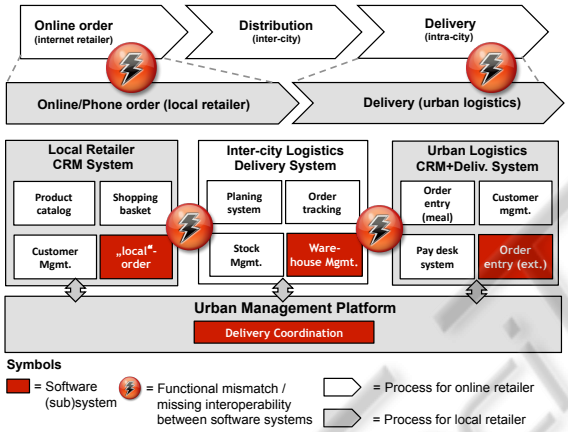

Figure 2: Interoperability problems between today's enterprise software systems when trying to implement an Urban Management Platform.

Synchronizing business models and architectures. The next generation of enterprise software shall help to overcome these difficulties. It will support the development of software that interacts smoothly across company borders and can be easily used from the participating parties' processes. There is a strict coherence between the comprehensive high level business process, a software module pool supporting different functional steps and the seamless integration of these within the landscape of the shareholders. In addition, new solutions are required to engineer BPM software components supporting the different process-shareholder's BPM phases.

Additional requirements (R) for business process modeling and architecture.* Building such enterprise software systems, however, imposes heavy functional and non-functional requirements: As we have seen, we cannot anticipate the final integration scenarios. Thus, the involved software systems have to be ready for unforeseen adaptations *(R1)*, in particular to be conducted by independent service providers *(R2)*. Extending these systems has to be allowed conveniently without jeopardizing core business-critical functionality originally designed into the system *(R3)*. As the integrity of the core functionality is vital, constant quality control has to be ensured as well as interoperability in form of unambiguous models *(R4)*. Above all, rapid business model prototyping has to be supported by software systems to allow for experimenting with new business models without compromising existing core business functions *(R5)*. The systems need to collect and expose data about their usage in real-time *(R6)* to provide for the necessary monitoring information.

We advocate the use of emergence and emergent behavior in tomorrow's enterprise software systems to meet these conflicting objectives.

3 BRINGING EMERGENCE TO ENTERPRISE SOFTWARE

Emergence enables complex systems consisting of autonomous entities to adapt to changing environments and expose new functionality that has not been explicitly designed into them beforehand. Emergent behavior refers to the successful combination of uncoordinated interactions by the different autonomous entities making up a system-of-systems (SoS). Emergence materializes if the interactions inside a system create an advantage for the system within a given context. With these new emergent abilities a system can now pursue objectives that would be too complex for a single entity to handle - leading to a change in scale and scope (Boardman and Sauser, 2006). In that, emergent systems are not only state-preserving (i.e. self-organizing and adaptive) but *proactive* as they are able to actively utilize changes in their environment to their own benefit (Fromm, 2004). Consequently, the concept of emergence has long since been subject to multiple areas of research spanning from biology to philosophy to mechanical engineering (Fromm, 2004; Stepney et al., 2006). In computer science it is currently most prominent in the context of autonomous systems (AC), organic computing (OC), and systems-of-systems (SoS) from the military domain (De Wolf and Holvoet, 2005; Boardman and Sauser, 2006; Huebscher and McCann, 2008; Würtz, 2008; Blair et al., 2011). Although Gartner featured emergent (or *middle-out*) architectures as an interesting new perspective for enterprise architectures in

2009[1] and 2010[2], enabling emergent behavior in enterprise software systems is currently not a prominent research topic.

Emergent behavior in software systems is typically leveraged using nature-inspired mechanisms such as evolutionary algorithms based on simulated natural selection. Nevertheless, this approach accepts unstable system-states or the evolvement of undesired behavior at times that are hard to predict (Fromm, 2004; Stepney et al., 2006). Engineering emergent behavior in the domain of enterprise software, however, has to take into account at least three additional constraints *(C*)*: Enterprise software is business critical and needs to be compliant to business and legal constraints at all times *(C1)*. Humans, equipped with natural adaptability and independence, are always directly or indirectly involved *(C2)*. Emergent behavior should lead to benefits on a technical or organizational level *(C3)*. With regard to these constraints and the requirements *(R1)-(R6)* presented before, we define Emergent Enterprise Software Systems (EESS) as:

> *Emergent Enterprise Software Systems:* Component-based software systems offering new value added services efficiently due to (semi-)automatic adaptation and self-organisation without violating critical core functionalities. This adaptation can be pursued by integration, combination and modification of emergent software-components from different vendors even if this use has not been intended initially. In addition, emergence can be accomplished by unanticipated use by humans. Quality attributes and functionalities are conserved or enhanced under changing conditions.

Using EESS represents a competitive advantage for all parties involved. First, EESS allow for quickly implementing new or changed business-relationships (c.f. §1), thus reducing time to market for new services and enabling even small companies to extend their portfolio of products and services for end customers easily. Second, EESS allow for small and middle enterprises (SMEs) to operate custom tailored but interoperable software systems inexpensively (e.g. Cloud-based (Frischbier and Petrov, 2010)). Third, ESS are easy to maintain as they are able to compensate changing requirements (semi-)automatically. These objectives, however, go far beyond the scope of isolated technological approaches currently pursued. Therefore we have to combine these approaches to enable emergence in enterprise software. Unfortunately, constrains *(C1)-(C3)* prevent us from directly applying most of the mechanisms used in other domains to enable emergent behavior in software systems.

[1]https://www.gartner.com/it/page.jsp?id=1124112

[2]https://www.gartner.com/it/page.jsp?id=1358913

4 ENGINEERING CHALLENGES

When designing, building and operating EESS we have to meet the special requirements *(R1)-(R6)* together with the constraints *(C1)-(C3)* of business software in general. We will discuss the arising challenges from two perspectives: 1) *architecture* and 2) *business process modeling & governance*.

4.1 Architectural Perspective

Supporting reactivity, interoperability and extensibility by third parties while preserving core business functionality (R1) - (R3). In nature, complex systems exposing emergent behavior are usually federations of interacting and cooperating subsystems. This has to be reflected in the structure of an EESS.

Distinguishing emergent components, systems and the fabric (R4),(R6). From an architectural perspective, we have to distinguish between a *component* with emergent capabilites and an *emergent system* exposing emergent behavior.

Figure 3 depicts the structure of an EESS. It shows how traditional pull-based functionality centered on persistent data is combined with push-based management of event flows in components. These components with emergent capabilities are then combined into an EESS by a fabric. Figure 3 also identifies the main levels of abstraction: data level (handling of information either as persistent data or non-persistent flows of event-objects), building block level (pull- or pushed-based building blocks to embody an emergent component), component level (components and subsystems with emergent capabilities composing EESS), and system level (complex software systems exhibiting emergent behavior).

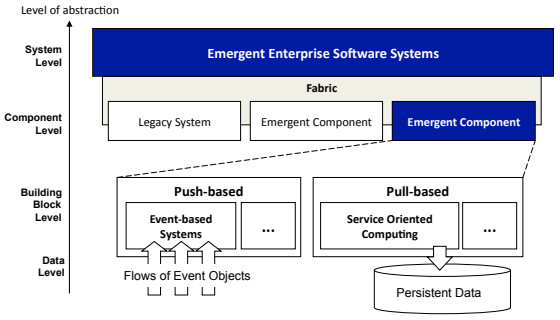

Figure 3: Architecture and building blocks of EESS.

Pull-based building blocks are the backbone of current software systems. They implement business processes on the software level by relying on persistent data and stable workflows identifying the participating subsystems. Service-oriented computing

(SOC) in particular enables reuse and modularization by encapsulating functionalities in implementation-independent services. Service-oriented concepts have already provided significant steps towards more flexible enterprise software systems within companies (Frischbier and Petrov, 2010). However, they are not yet ready for unanticipated changes across company borders and do not yet provide the self-monitoring capabilities needed (Frischbier et al., 2011).

Therefore we advocate to combine pull-based building blocks with *push-based* building blocks operating on streams of information. In contrast to pull-based building blocks, event-based systems (EBS) do not rely on persistent data but work on continuous flows of event objects. Event objects represent information on meaningful changes in the environment disseminated by event-producers and consumed by event-consumers. With the information itself defining the participating subsystems, event-driven building blocks easily adjust to changes in the environment. Using complex event processing (CEP) and rule engines to react according to given (or learned) rules, event-based systems transform patterns of information into functionality. Today, event-based systems are primarily used in the context of business intelligence to quickly gain knowledge and react on it (Castellanos et al., 2010; Buchmann et al., 2010).

The *fabric* acts as a coordinator and handles the information flow between emergent components, emergent systems and legacy systems. Therefore, it is able to: i) balance components' different expectations regarding quality of service (QoS) and quality of information (QoI) in a distributed way; ii) handle different semantics of the exchanged information; and iii) allow for anonymous introspection and monitoring of components (cf. *(R6)*).

The two main challenges from an architectural point of view are: i) integrating pull- and push-based building blocks into emergent components; ii) enabling the fabric to connect emergent components and legacy systems while allowing for minimal-inversive introspection and seamless monitoring at runtime even across company borders.

4.2 BPM & Governance Perspective

Supporting rapid business model prototyping while keeping models synchronized (R1), (R3)-(R5). Engineering EESS is strictly correlated with BPM. Having a closer look at the phases and overlying processes of BPM shows that today the life-cycle phases (i.e., process strategy, design, implementation, execution and controlling) are supported individually by software tools using distinct methods and technologies. Integrating them is difficult if not impossible. Engineering EESS, however, requires a *seamless process of process management* where strategy-defining models and methods are aligned with those for designing business processes. Thus it is of vital importance to procure interoperability within the BPM life-cycle by developing integrated models, methods and transformation tools to allow for emergence between the BPM software components. Already implemented digital process models must be easily transform- and (re)implementable when new emergent components are added to the EESS. For this purpose, EESS need modular and flexible capabilities to track the dynamically implemented processes even if they span company borders. Business processes, however, are the core assets of any company and none of them is interested in sharing valuable process experience with other companies without compensation. Thus, software providers have to act as mediators between different business processes - not only inside a business line but also between different industries.

EESS need to be open for third-party extensions at a larger scale than today while consumers and providers may not necessarily know about each other beforehand. Therefore, an *integrated governance and compliance model* needs to span all existing types of used software building blocks (e.g., business processes, services in SOC, or entities in EBS). As we cannot assume a central authority to control the evolution of EESS, governance and compliance models have to: i) allow all parties to integrate services into an open EESS without impairing the quality of existing configurations; ii) prevent dependencies on external processes to compromise a constant level of quality for information and services.

Linking runtime and design-time information using feedback (R1), (R3)-(R6). EESS are continuously evolving as they react to changes in their environment. Thus, development has to take uncertainty into account as neither the final relationships to other entities nor the final requirements may be known at design-time. This calls for a strong *feedback loop* between runtime and design-time: The monitoring of systems and components at runtime that allows for a comparison between de-facto and expected behavior. Thus, methodology has to focus on selecting, adapting, extending, composing and testing emergent components to form emergent systems based on that feedback without violating required levels of quality. Especially components being partly deployed and systems in different life-cycle stages have to be considered. Typically, changes to EESS require a detailed picture and governance process on the application landscape

including involved organizations, users, roles, or resources used. Anticipating system usage and detailed planning in advance has to be at the core of the development and runtime life-cycles of an EESS.

Enabling life-cycle-wide self-adaptation to support change management (R2), (R3), (R6). The creation and roll-out of new services/processes or new versions due to changes in functionality, performance, or pricing will be simplified, fastened, and even be (partly) automated with EESS. As a consequence, life-cycle changes like publishing, replacement or retirement will happen more frequently. While change management tasks have usually been executed as part of planning and testing stages before a roll-out of new functionality they will blend into *continuous monitoring and enforcement tasks at runtime*.

In sum, the three main challenges from a BPM & governance point of view are: i) integrating method- and tool-support for the various BPM life-cycle phases; ii) providing holistic governance and compliance methods across distributed organizations, business processes and components; iii) taking dynamics and unpredictability into account.

5 OUTLOOK & FUTURE WORK

Enterprise software systems still lack the interoperability and flexibility necessary for companies to gain competitive advantage in fast changing markets. In this position paper we sketched out a solution based on the concept of emergence and identified the main challenges related to architecture, business process modeling (BPM) and governance. Future work in this area will focus short term on implementing first solutions in current industry-strength software systems and academic prototypes. Longterm, the results of academic research and industrial expertise will be combined into Emergent Enterprise Software Systems (EESS) as envisioned here.

We tackle the arising research challenges within the research project EMERGENT in four work packages: *Interoperability* focuses on new concepts and techniques to provide interoperability in ESS. *Adaptivity* works on dynamic adaptation of heterogeneous information infrastructures across enterprises and the optimization of (business) processes based on the available resources or current usage. *Usability and User Context* explores how to interact with emergent software in different and varying user contexts. *Security* focuses on fundamental security-related concepts and technologies.

ACKNOWLEDGEMENTS

The work presented in this paper was performed in the context of the Software-Cluster project EMERGENT. It was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IC10S01. The authors assume responsibility for the content.

REFERENCES

Blair, G., Bennaceur, A., Georgantas, N., Grace, P., Issarny, V., Nundloll, V., and Paolucci, M. (2011). The role of ontologies in emergent middleware: Supporting interoperability in complex distributed systems. In *Middleware 2011*.

Boardman, J. and Sauser, B. (2006). System of systems-the meaning of of. In *SYSOE 2006*.

Buchmann, A., Pfohl, H., Appel, S., Freudenreich, T., Frischbier, S., Petrov, I., and Zuber, C. (2010). Event-Driven services: Integrating production, logistics and transportation. In *SOC-LOG 2010*.

Castellanos, M., Dayal, U., and Hsu, M. (2010). *Live Business Intelligence for the Real-Time Enterprise*, volume 6462 of *LNCS*, pages 325–336. Springer.

De Wolf, T. and Holvoet, T. (2005). *Emergence versus self-organisation: Different concepts but promising when combined*, volume 3464 of *LNCS*, pages 77–91. Springer.

Freudenreich, T., Appel, S., Frischbier, S., and Buchmann, A. (2012). ACTrESS - automatic context transformation in event-based software systems. In *DEBS'2012*.

Frischbier, S., Buchmann, A., and Pütz, D. (2011). FIT for SOA? Introducing the F.I.T. - metric to optimize the availability of service oriented architectures. In *CSDM 2011*.

Frischbier, S. and Petrov, I. (2010). *Aspects of Data-Intensive Cloud Computing*, volume 6462 of *LNCS*, pages 57–77. Springer.

Fromm, J. (2004). *The Emergence of Complexity*. Kassel University Press.

Huebscher, M. C. and McCann, J. A. (2008). A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40:7:1–7:28.

Stepney, S., Polack, F., and Turner, H. (2006). Engineering emergence. In *ICECCS 2006*.

Würtz, R. (2008). *Organic computing*. Springer.