

Towards a Model-based Service Integration Framework for Extensible Enterprise Systems

Matthias Allgairer, Markus Heller*, Martin Weidner*

SAP Research, Karlsruhe

1 Motivation and Problem Description

In the vision of an *Internet of Services* (Janiesch et al. 2008, S. 71-75), services will become tradable similar to manufactured goods. Organizations dynamically interact as service consumers and service providers making use of a service marketplace to design, offer and consume services. With the emergence of such *service ecosystems* (Barros and Dumas 2006, S. 31-37), the simplified consumption of services (Papazoglou 2007, S. 5-6) becomes a key challenge for enterprise organizations. The direct integration of services (offered via a service marketplace) into standard business applications running within enterprise systems (e.g. ERP systems) is most promising as these systems reflect the core business processes of an organization and can flexibly be enriched with complementary services.

From the perspective of an enterprise system two different service integration scenarios can be distinguished (Figure 1).

In the *first scenario* the enterprise system is extended with a service using a pre-defined service interface that has explicitly been *foreseen* by the enterprise system provider when the system was shipped. Examples are (de-facto) standard interfaces for Business-to-Business (B2B) or Application-to-Application (A2A) integration scenarios known from the area of Enterprise Application Integration (EAI).

In the *second scenario* the enterprise system is extended with a service using a service interface that has *not been foreseen* by the enterprise system provider when the system was shipped. Before the service can be used core business application(s) running within the enterprise system need to be adapted/extended on the affected application layers, e.g. by adding new UI elements (presentation layer), adding a new process step (business process layer) or extending a business object with a new field (business object layer). Examples are services innovated by service providers in dynamically evolving service ecosystems. In both scenarios structural-

* The work presented in this paper is embedded into THESEUS/TEXO which is funded by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ07012. The authors take the responsibility of the contents.

and/or behavioural mismatches between the service interfaces of the enterprise system and the service provider are handled by *service mediation* components.

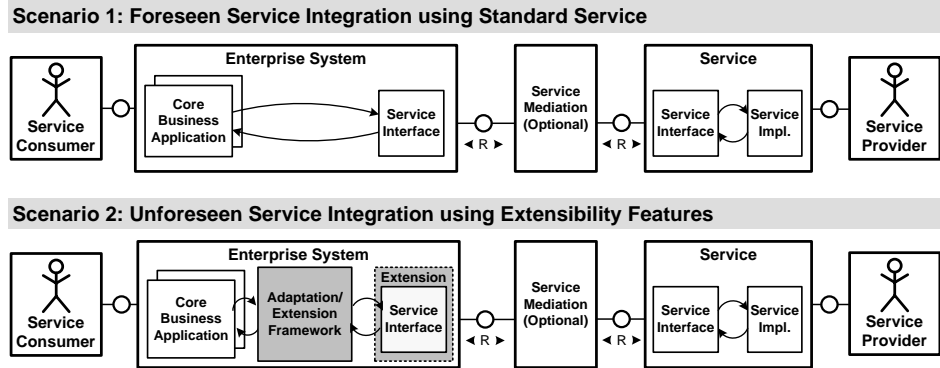


Figure 1: Service Integration Scenarios for Enterprise Systems

While multiple works exist that address the service mediation problem (scenario 1/2), to the best of the authors' knowledge, no systematic engineering methodology exists for the integration of unforeseen services (scenario 2) that will be the focus of our research work. Typically enterprise systems only provide proprietary adaptation and extension techniques with a low level of abstraction often requiring (i) code modifications of the core business applications with a negative impact on software-lifecycle-management as well as (ii) deep technical programming skills¹.

In the last years *Software-as-a-Service (SaaS)* delivery models of enterprise systems gained momentum in the enterprise software market (Lo et al. 2009, S. 10). As SaaS subscribers do not employ specialized integration experts a more efficient integration of services becomes a major requirement. We envision an approach that (i) allows enterprise system providers to ship standard business applications and (ii) at the same time offers dedicated extensibility features that allows partners in a service ecosystem to seamlessly integrate new services at a later stage in the software-lifecycle. Instead of programming and installing new components into a dedicated host environment, as for example known from plug-in technologies (e.g. Eclipse, cf. (Birsan 2005, S.40-42)), our work focuses on service integration.

We will address this problem with the proposition of a *Service Integration Framework* as part of the THESEUS/TEXO². Our work applies the design science research methodology (Hevner et al. 2004, S. 75-105). This paper is structured as follows: In Section 2 a motivating scenario is introduced; Section 3 outlines the basic requirements for the framework. In Section 4 the main concepts of the framework are presented and in Section 5 a first running prototype is described

¹ e.g. SAP AG, ABAP Enhancement Framework, <http://help.sap.com>, visited 10-09-2009

² <http://theseus-programm.de/en-us/theseus-application-scenarios/texto/>, visited 10-09-2009

that demonstrates the integration of a service into a user interface. Section 6 briefly describes related work and Section 7 gives an outlook on future work.

2 Motivating Scenario

The following scenario from the automotive sector illustrates the envisioned integration of tradable services into enterprise systems. Due to legal changes in export guidelines, a *manufacturer of car seats* has to certify his products to guarantee that materials used within a car seat comply with ecological laws. A *service provider* offers a service on the *Service Marketplace* that allows the calculation of eco values for products including certification. The company runs an enterprise system including a Product-Lifecycle-Management (PLM) module that supports the company in the design process of car seats (Screenshot in Figure 5 on page 1532). The core version of this business application does not support the calculation of eco values for a given bill of material. A product designer of the company accesses the Service Marketplace directly from within his enterprise system and searches for services that provide the missing functionality. He receives a list of matching services from various service providers certified for his enterprise system. According to his working context the designer selects a service called “*Eco-Calculator*” and buys it on the marketplace. Subsequently the service is automatically integrated into the core business application without running a manual integration project: the user interface of the core business application is extended with (1) an additional table column (“Eco Value”) in the product components table, (2) an additional button (“Calculate Eco Value”) and (3) an additional field indicating the total eco value for the car seat (“Entire Eco Value”). After the service is integrated into the consumer application, the service can be used. If the total eco value fulfils the legal requirements, a certificate is generated and passed to the consumer application.

3 Requirements

In the previous section, a motivating scenario has been presented to illustrate the user’s perspective on service integration within enterprise systems. In the given context of service marketplaces seamless and less-complex service consumption is needed. Today, three *main deficits* can be observed in integration projects when an enterprise system is extended by additional functionality or services: (1) Often much *manual effort* (programming) is needed in major tasks to extend an enterprise system. (2) Typically enterprise systems only provide proprietary adaptation and extension techniques with a *low level of abstraction* (e.g. proprietary code-level interfaces where additional integration code can be plugged in). (3) It is often in the *responsibility of the human integration expert* to identify and implement an optimal integration solution, to ensure that all needed integration details for the chosen integration alternative are provided and e.g. nothing is forgotten.

The following *research requirements* can be derived from the motivating scenario and the above observations for the development of a *service integration framework*:

- (1) Develop a *model-based integration approach* to enable an integrator to model (or design) the relevant integration aspects on a higher abstraction level (than e.g. on implementation-level). For example, model checking, simulation, and design recommendation support can be offered on top of it.
- (2) Enable a *controlled extensibility* modelling insofar as only a proven set of integration operations can be performed. This set of actions should capture best-practice knowledge about frequent and relevant integration tasks.
- (3) Develop a *uniform modelling approach* to enable the integrator to design the extension of an application on multiple application layers.
- (4) Allow for the *explicit definition* and *re-use* of best-practice integration knowledge in the course of an integration project.

We will address these requirements within the Service Integration Framework.

4 Service Integration Framework

This section presents the conceptualization of the Service Integration Framework. In Subsection 4.1 the Service Marketplace scenario with relevant stakeholders and their interactions is described. Subsection 4.2 introduces the main concepts of our framework. The layered modelling approach is explained in subsection 4.3 with a first sketch of a meta model as the central foundation of the envisioned solution. Subsection 4.4 focuses on a context notion and adaptation patterns.

4.1 Service Ecosystem - Roles and Interactions for Service Integration

The *Enterprise System Provider* (Figure 2, lane 1) delivers standard business applications running within enterprise systems. At the same time these core business applications provide the capability to be flexibly adapted/extended at a later stage in the software-lifecycle. Therefore a description on how the core business applications can be extended on the relevant application layers is delivered. A *Service Provider* (lane 3) develops and tests a service, independent of the concrete service consumer environment. The *Service Integrator* (lane 2) is responsible for providing an integration solution for the combination of (i) a core business application running within an enterprise system and (ii) a service, leveraging the extensibility features of the core enterprise system. The integration solution is tested (and optionally certified by the enterprise system provider) and published on the *Service Marketplace* (lane 4) where it can be offered to a potential high number of service consumer. A *Service Consumer* (lane 5) can search for services offered on the service marketplace to extend his core business application provided by the enterprise system provider. Instead of purchasing a plain service description and manually integrating the service into his core business application(s) the service consumer can select the integ-

ration solution offered by the service integrator. This solution allows the automatic adaptation/extension the core business application(s) so that the service is fully integrated into the enterprise system and ready for usage.

After the usage phase the service marketplace charges the service consumer for the service usage and the service provider for the brokering of the service.

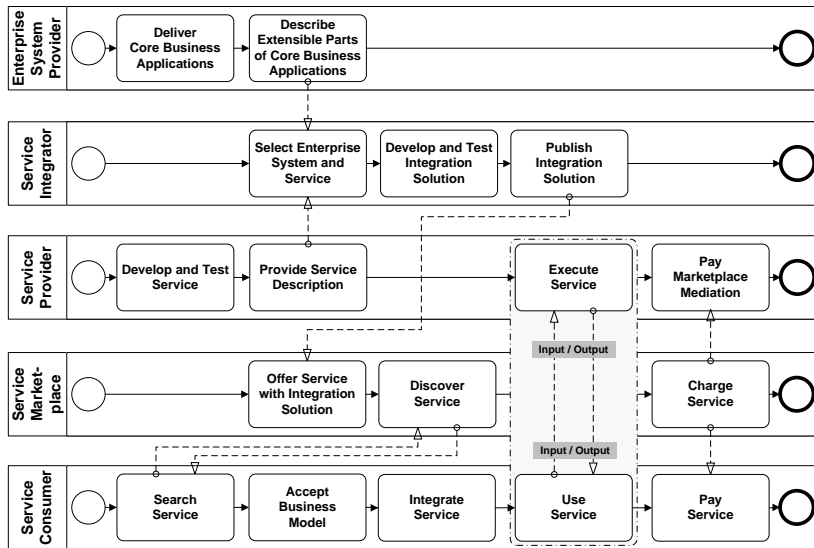


Figure 2: Service Ecosystem – Roles and Interactions

4.2 Service Integration Modelling - Core Concepts

All information needed for service integration is captured within a set of *descriptions* described below (Figure 3). Three different *descriptions* are created and maintained by the responsible roles as follows:

- (1) The *Application Extensibility Description* (left) represents a model of the core business application’s extensibility capabilities and is maintained by the enterprise system provider. It contains all possible extension points where the application can be extended or adapted. Such *extension points* model the offered *extensibility features* of the application on top of the underlying enterprise system. For example, extension points denote places in the application that can be used to add a process step to a core process. Alternatively, extension points for the user interface allow the addition of a new UI element, and so on.
- (2) The *Service Description* (right) models the service’s capabilities with respect to multiple aspects and is created by the service provider. Among other information, the service description contains relevant information for the integration of the service into core business applications (such as service operations with input and output interfaces, supported data types, messaging choreographies, offered default UI descriptions).

- (3) An *Integration Description* (bottom) is developed by the service integrator and contains a set of relationships between elements in the enterprise extensibility description and elements the service description. For this purpose, an *integration relationship* or *integration link* is established between both elements³. The whole integration description is the outcome of an integration design process based on the two inputs (1) and (2). The integration description specifies all steps to achieve the desired integration and can additionally reference any combinations of software artefacts needed for the integration runtime system.

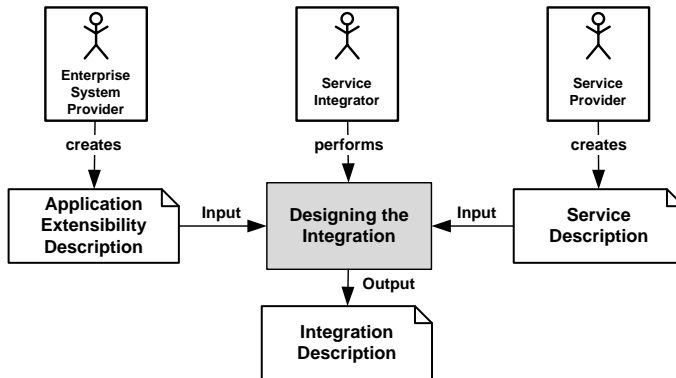


Figure 3: Inputs and Output for Integration Design

4.3 Layered Modelling Framework

Following a model-based approach, all descriptions introduced in the preceding section are based on a common conceptual basis formalized in a meta model:

- (1) The overall *adaptation meta model* defines all concepts for application extensibility descriptions, service descriptions and integration descriptions. It describes identified best-practice integration knowledge on a *type-level* modelling layer (e.g. how and with which constraints to relate service capabilities to the application extensibility capabilities).
- (2) *Integration models* represent concrete application extensibility descriptions, service descriptions, or integration descriptions on an *instance-level* modelling layer. Integration models are described with the concepts in the adaptation meta model to detail all information for a concrete integration (e.g. for the integration of the service “Eco-Calculator” from Subsection 2).

Figure 4 illustrates both modelling layers with a cut-out of relevant elements of the adaptation meta model (upper part) and instances of concrete integration mod-

³ Precisely, an integration link connects a concrete extension point and a concrete *connector*. The connector concept is described more in more detail in Section 4.3.

els (lower part). The *adaptation meta model* (upper part of Figure 4) is partitioned into a core meta model part at the top and several specific refinement parts below.

The *core part* of the meta model defines all core conceptual entities and their relationships. For example, as explained below we separate between five central modelling areas for modelling different part of a core business application, the defined extension points, integration links (as explained above), connectors (explained below), and the relevant parts of a service. For all areas, all modelling elements of an area inherit from corresponding root model entities: Core Application Entity, Core Extension Point Entity, Core Integration Link Entity, Core Link Connector Entity, Core Service Entity.

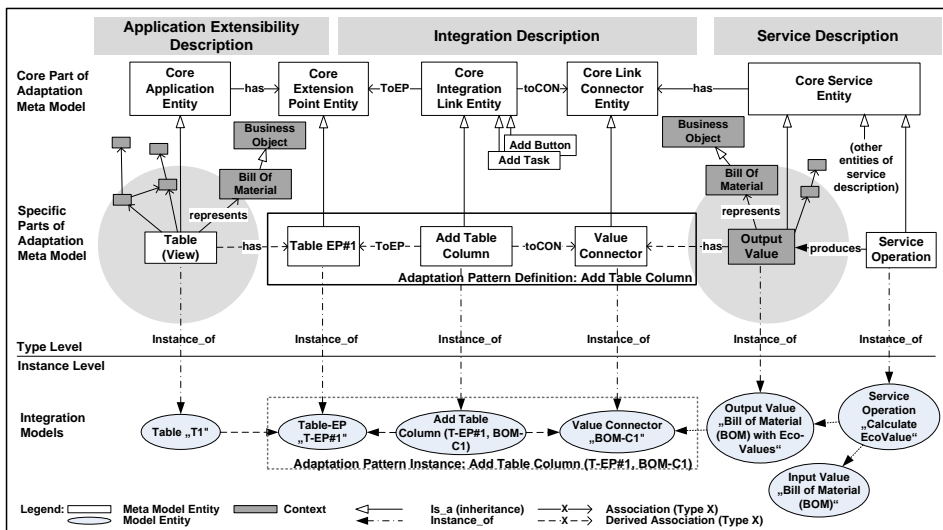


Figure 4: Meta Model and Integration Models

Similarly, all relevant relationships between these entities are part of the core meta model part. For example, the relationship *has* is modelled between Core Application Entity and Core Extension Point Entity to denote that each entity of a core business application can have (null or many) assigned extension points and so on. The central modelling element Core Integration Link Entity models a direct connection between an extension point (Core Extension Point Entity) and a connector (Core Link Connector Entity). The latter entity is used as a surrogate to reference to one or more elements of a service description (model) indirectly. Not all possible combinations of extension points and connectors are allowed. Instead, in the sense of a required *controlled extensibility* (see requirement 2), it is formalized within the meta model definition for the integration link (Core Integration Link Entity) which combinations are allowed (while all other combinations are forbidden).

The *specific parts* of the meta model describe some additional aspects of integration knowledge without a general meaning and therefore they are not in the core meta

model part. For example, the figure shows extensibility capabilities of tables (**Table**) that are shown in a user interface of the core business application and one (of many possible) extension point (e.g. **Table EP#1** (table column addition)) for a UI table are depicted. The relationships of specific elements (e.g. **Table**) to elements in the core part (e.g. **Core Application Entity**) is given by inheritance relationships **instance_of** (other kinds of associations are possible). Because of the inheritance relationships, the same relationships do not have to be modelled repeatedly in the specific part and can be *derived* instead⁴. The extension point **Table #EP1** (table column addition) is linked with a connector representing a certain value (**Value Connector**) via a **Core Integration Link Entity** of type **Add Table Column**. Similar integration relationships can be modelled for other combinations of extension points and connector entities.

The lower part of Figure 4 shows a cut-out of related instance-level integration models for application extensibility description, service description and integration description. For illustration, the models are shown for the example of the service “Eco-Calculator” into the described PLM business application mentioned in the motivating scenario (see Section 2). The integration model for a service description (shown to the right) contains some important modelling elements for a (web) service operation “calculateEcoValue” with one input and one output parameter. The left part of the figure shows a cut-out of the integration model for PLM application extensibility description with one specific selected table in the user interface and one of the extension points allowed for this table: this extension point is for adding a column to the given table. The middle part of the figure shows the integration model for the integration description with an application of the adaptation pattern **Add Table Column** with its associations to the relevant entities in the integration models for the application extensibility description and the service description.

With this example the pattern-based modelling approach on the presentation layer is introduced. In future versions we will extend this concept to address the adaptation of the business process layer in the same way.

4.4 Context Orientation and Adaptation Patterns

On both, the meta model and the integration model layer, a *context* for model entities consists of connected model entities and their relationships. For example, in the adaptation meta model **Table** and **Value Connector** both are linked via associations **represents** to element **Bill Of Material**, which is in turn an instance of a **Business Object** meta model element (the context is indicated as a grey circle in the figure). This context realizes a higher-value annotation and allows for matchmaking in the meta model (and accordingly on the instance level). For

⁴ For example, the relationship **Has** between **Table** and **Table #EP1** is implicitly already modelled in the core part of the meta model which avoids modelling the relationship **Has** between **Table** and **Table EP#1** again.

example, possible candidates of extension points for the integration can be discovered among all extension points based on given contextual dependencies that have to be matched (e.g. referring to instances of a certain Business Object).

Adaptation patterns are introduced to group common patterns of model elements (and their relationships) into a pattern template for later re-use of the conserved model structure. An adaptation pattern defines a certain association structure between an integration link and one or many other associated entities of the meta model. For example, an adaptation pattern **Add Table Column** comprises the (specific, table-related) meta model entities **Add Table Column** (a specific table-related integration link), **Table** and **Value Connector**. A large set of adaptation patterns can be defined to model very basic or elementary integration steps. Likewise, adaptation patterns can be defined on the process layer (e.g. patterns like **Add Process Step**, **AddControllflow** or on the service / business object layer (e.g. patterns like **Add Business Object Attribute**). Another adaptation pattern **Add Data Mapping** can address the data mapping between data passed between the user interface of the core business application and a called web service forth and back. For the Eco-Calculator example, bill of material data from the UI is mapped to an input message of the web service “EcoCalculator” and the output message is mapped to further UI elements. Adaptation patterns serve to help realizing controlled extensibility (requirement 2) and to help in explicitly modelling and defining best-practice patterns (requirement 4). The proposed conceptual framework for service integration as detailed in this section addresses all requirements 1 to 4 from Section 3.

5 Demonstration Prototype

To demonstrate the proposed approach we have implemented a running prototype for the Eco-Calculator scenario introduced in Section 2. The left part of Figure 5 shows the dialog that is used by the service consumer to search for applicable services on the service marketplace. When the user selects a service it is automatically integrated into the business application. The right part of Figure 5 shows the PLM business application that has been implemented based on the Microsoft Silverlight framework⁵. During the integration of the service the core business application is extended on the user interface layer with (1) an additional table column, (2) an additional button and (3) an additional field.

⁵ <http://www.microsoft.com/Silverlight/>, visited 10-09-2009

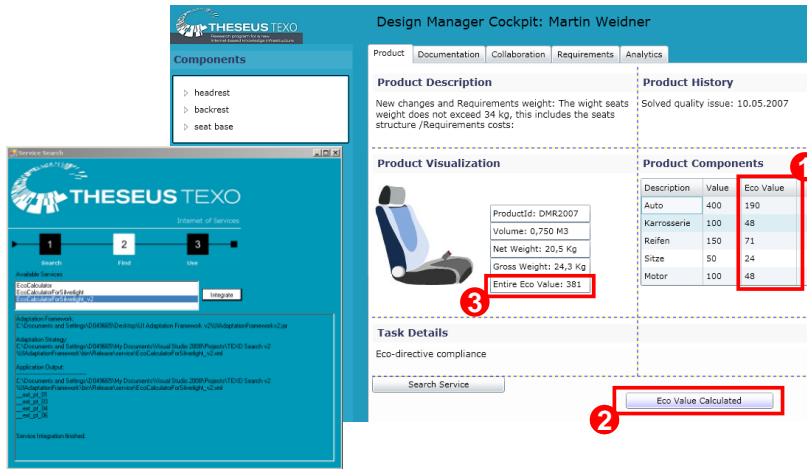


Figure 5: Prototype of Business Application

In Figure 6 the architecture of the prototype is shown. The *application extensibility description* (left part) defines three instances of the type Core Application Entity representing the adaptable parts of the core business application. Each of these models provides a dedicated extension point of the type Core Extension Point Entity: T-EP#1 allows the extension of table T1 with an additional table column, BP-EP#2 allows the extension of the button panel BP1 with an additional button and P-EP#3 allows the extension of the panel P1 with another field.

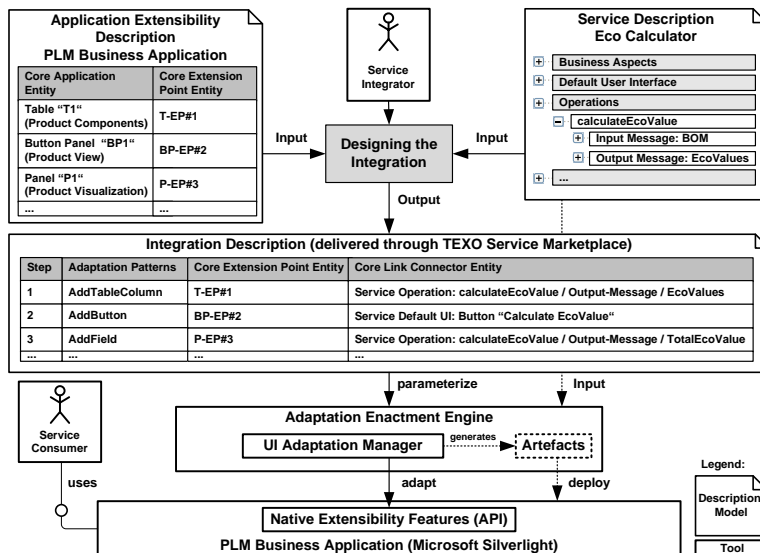


Figure 6: Architecture of Prototype implemented in Microsoft Silverlight

The *service description* (right part) with its different aspects is shown, modelled as instances of type **Core Service Entity**. Based on these models the service integrator defines the *integration description* (middle part of Figure 6) that instantiates three adaptation patterns for the required extensions: **AddTableColumn**, **AddButton**, **AddField**. The *Adaptation Enactment Engine* (implemented in Java) is parameterized by the integration description and forwards it to the *UI Adaptation Manager* that actually adapts the PLM business application by reusing its native extensibility features (e.g. APIs) provided by the Microsoft Silverlight framework⁶.

6 Related Work

As outlined in Section 1 our work is related to existing approaches from the areas of *B2B Integration*, e.g. (Bussler 2003) and *Enterprise Application Integration (EAI)* e.g. (Hoppe and Woolf 2003). Instead of focusing on bridging structural- and/or behavioural mismatches between service interfaces using an Enterprise Services Bus (ESB) our work concentrates on the controlled extensibility of an enterprise system for unforeseen service integration, cf. (Studer et al. 2007, S. 259-279).

Different related work is available in the field of *adaptive software systems*, however with the majority of work focusing on self-adaptive systems in the domain of mobile and pervasive computing (cf. MADAM/MUSIC research projects⁷).

In the area of *component-based software engineering* our work is related to the concept of plug-in technologies that allow the development and installation of components into a core application framework (Birsan 2005, S.40-46). Plug-in techniques have also been used in combination with a software product line suite to support runtime adaptation of an ERP system (Wolfinger et al. 2008, S.21-25). In contrast to these plug-in approaches we concentrate on service integration, not on component integration, cf. (Overhage and Turowski 2007, S. 4-8).

For the extension of the presentation-layer our work is related to approaches from the field of *Adaptive User Interfaces* mainly focusing on the abstract definition of user interfaces (e.g. XIML or UIML⁸). For the extension of the business process layer of an enterprise system (future work) our work is related to the area of *Adaptive Process Management*, mainly with respect to process flexibility patterns cf. (Mulyar et al. 2008, S. 95; Weber et al. 2008, S.438-466).

7 Conclusion and Future Work

We have presented a model-based framework to achieve the unforeseen integration of services into extensible enterprise systems. A prototype for presentation

⁶ Optionally platform-specific artefacts will be generated and deployed by the UI Adaptation Manager
⁷ <http://www.ist-madam.org>, <http://www.ist-music.eu>, visited 10-09-2009

⁸ <http://www.ximl.org>, <http://www.uiml.org>, visited 10-09-2009

layer integration of services into core business applications demonstrates our approach. The current prototype is restricted to the extension of the presentation layer. In future versions we plan to support the adaptation on the business process layer in the same way as this becomes particularly relevant in enterprise context.

References

- Barros A, Dumas M (2006) The Rise of Web Service Ecosystems. *IT Professional* 8 (5): 31-37.
- Birsan D (2005) On Plug-ins and Extensible Architectures. *Queue* 3 (2): 40-46.
- Bussler C (2003) *B2B Integration: Concepts and Architecture*. Springer, Berlin.
- Hevner AR, March ST, Park J, Ram S (2004) Design Science in Information Systems Research. *MIS Quarterly* 28 (1): 75-105.
- Hohpe G, Woolf B (2003) *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, Boston.
- Janiesch C, Ruggaber R, Sure Y (2008) Eine Infrastruktur für das Internet der Dienste. *HMD - Praxis der Wirtschaftsinformatik* 45 (261): 71-79.
- Lo H, Wang R, Garbini, JP (2009) *The State of Enterprise Software 2009*. Forrester Research, Cambridge.
- Mulyar N, van der Aalst WMP, Russell N (2008) Process flexibility patterns. BETA Working Paper Series, WP 251, Eindhoven University of Technology.
- Overhage S, Turowski K (2007) Serviceorientierte Architekturen - Konzept und methodische Herausforderungen. In: Nissen V, Petsch M, Schorcht H (Hrsg) *Service-orientierte Architekturen. Chancen und Herausforderungen bei der Flexibilisierung und Integration von Unternehmensprozessen*. Deutscher Universitätsverlag, Wiesbaden.
- Papazoglou MP (2007) *Web Services: Principles and Technology*. Pearson - Prentice Hall, Upper Saddle River.
- Studer R, Grimm S, Abecker A (2007) *Semantic Web Services - Concepts, Technologies and Applications*. Springer, Berlin.
- Weber B, Reichert M, Rinderle-Ma S (2008) Change patterns and change support features - Enhancing flexibility in process-aware information systems. *DKE* 66 (3): 438-466.
- Wolfinger R, Reiter S, Dhungana D, Grünbacher P, Prähofer H (2008) Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques. In: 7th IEEE International Conference on Composition-Based Software Systems (ICCBSS'08), Madrid.