Maximizing the Business Value of Networked Embedded Systems through Process-Level Integration into Enterprise Software

Patrik Spieß, Stamatis Karnouskos
SAP Research, Vincenz-Priessnitz-Strasse 1, D-76131 Karlsruhe, Germany
patrik.spiess@sap.com;stamatis.karnouskos@sap.com

Abstract

Building applications on resource-constrained networked embedded systems (the front-end) such as automation devices, (wireless) sensor networks etc, and integrating them into business processes of an enterprise (the back-end) is a complex, challenging task that has to be repeated for each combination of back-end and front-end application. We argue that using modelling on top of service-oriented architecture in networked embedded systems simplifies application development and enables easy enterprise-wide integration by using executable descriptions of cross-system business processes. We derive requirements for networked embedded systems, the communication points with enterprise software to support this model-driven software design, and directions for their implementation.

Keywords: Pervasive Computing, Context Technology, Business Process Management, Automatic Process Optimization, Back-end Front-end Integration.

1. Introduction

In the past, motivated by the classical perspective of Taylorism, businesses were organized along corporate functions. This approach proved too inflexible for the competitive situations today's enterprises find themselves in where survival requires constant change through innovation. Experts from the field of business studies claimed that organizing enterprises along their processes rather than functions allows for tighter integration and more flexibility (see Figure 1). Corporate IT software followed that trend.

IT-supported business processes have proven to be powerful tools for the modelling and execution of applications as computer-supported workflows like they appear within an organization (providing efficient application integration) or span across multiple organizations (serving as B2B integration tools). To support this, modern business software is engineered (and legacy business software is re-engineered) to offer high-level service interfaces that are universally accessible,

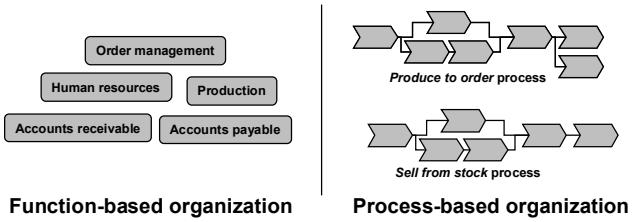

Figure 1. Organization along functions / processes

independently from programming languages or operating systems. This goal is primarily reached by the use of web services. Hence, business processes form the "glue" for orchestrating the offered services, defining meaningful rules for instantiation and termination of the process, sequences of service calls, result processing, and error compensation strategies.

Networked embedded systems comprise a broad range of technologies, like wireless sensor (and actor) networks, ubiquitous computing technology, intelligent production machines, industrial automation etc. As devices of all of these classes have become more powerful with regard to computing power, memory, and communication, they are starting to be built with the goal to offer their functionality as one or more services for consumption by other devices or services (SOA-ready devices). Therefore, we have a paradigm shift as these devices can offer more advanced access to their functionality and even host and execute business intelligence, therefore effectively providing the building blocks of a service-oriented architecture, just as contemporary enterprise-scale IT systems do.

This allows for an obvious, frictionless way of integrating those embedded services into corporate business processes. A business process expert would model a business process using both embedded services and the ones offered by the back-end. This cross-technology business process will be executed anywhere within the hybrid system consisting of back-end and front-end, possibly completely in the networked embedded subsystem. The new flexibility allows the business process expert to place its intelligence where it is needed, close to the point of action and distribute it over several layers among the devices and the enterprise systems. This capability enables the creation of more flexible and sophisticated business processes.

2. Back-end and Front-End Infrastructure

2.1. Back-end: Enterprise-Scale IT Systems

Contemporary IT systems introduce various levels of abstraction. At the very highest level, applications are modelled on top of complex frameworks abstracting from operating systems and offering integrated management of data communication and persistence. Although there exist many other application-layer paradigms for collaboration between systems, there is a trend towards using the web service family of standards (WS-*) as the only component model and collaboration paradigm. The WS-* standards [8] solve many aspects of communication like addressing, discovery, eventing etc. at the application layer. On the lower levels, all these systems are based on a common transport layer (e.g. TCP, UDP etc) over an IP based networking environment that is realized over (wireless) Ethernet. Overall, the back-end systems follow uniform design principles, facilitating integration and collaboration.

2.2. Front-end: Networked Embedded Systems

In the domain of networked embedded systems, the level of standardization is much lower than in the back-end. Heterogeneity starts at the lowest hardware levels, where many different, incompatible processor and memory architectures are used, and goes up to the upper layers of the communication stack where communication between devices is still mostly proprietary or follows (usually partially) one of the many available standards. Communication standards are mostly domain-specific, meaning that vendors of e.g. home automation and industrial automation are likely to use different signals, codes, protocols, and data representations. This is due to the lack of resources on embedded devices, which let manufacturers optimize and tailor their software for efficiency. Incompatibility led to market fragmentation and hence inability to support collaboration of multiple solutions on the device level. This diversity of networked embedded systems makes it hard to integrate networked embedded devices both with each other and with back-end enterprise systems. However, with ever increasing processing power and memory available in embedded hardware, it is foreseen that in the mid-term a consolidation of communication paradigms will take place, just as it occurred in conventional IT systems, that has lead to the current application stack (i.e. business processes using web services accessible over HTTP over TCP over IP). Service-oriented architectures, using web services or equivalent technology that allows for seamless integration in an infrastructure are a promising centre of gravity for the expected consolidation of the front-end.

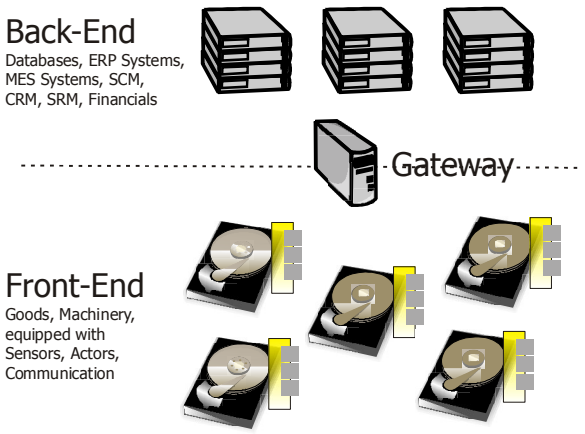

Figure 2. Back-end and Front-end topology.

2.3. Challenge: Integrated Business Processes

The heterogeneity of the front-end and the generally different paradigms of communication make integration of font-end and back-end systems difficult. To lower the required effort we propose to use executable, cross-system business process descriptions. To provide an environment supporting such descriptions, both the capabilities of the networked embedded systems and current process description languages have to be extended. As described in section 2.2, networked embedded systems will have to offer their functionality as a set of services. Each network node will host a set of service instances. For instance, each sensor node that is capable of measuring pressure must host an instance of a pressure measurement service. The signature of that service, meaning the available *actions* (the SOA-equivalent to what is called *methods* in the object-oriented world, including parameter types) and generated events (also called *notifications*) would be uniform for each service instance.

This approach would not break any interaction patterns used in the communication within networked embedded systems. They communicate usually in three ways: i) generating events, ii) making use of other (local and remote) services and iii) offering services; all of which are supported within a service-oriented architecture. There is some criticism about using RPC-style interaction in ubiquitous computing [9]; however, it only applies to the use of blocking, synchronous RPC calls neglecting the fact that services also allow for one-way messaging.

In theory, such service-oriented networked embedded systems could offer their services as web services, which use SOAP [5] over HTTP. While this is the most interoperable way of relaying information, it is highly inefficient regarding the scalability of communication, especially if we consider that in the envisioned “Internet of Things” [6] there will be millions of devices. In an infrastructure highly populated with networked devices,

Figure 3. Proposed process lifecycle

this communication flow would lead to high-utilization of resources at the back-end systems as well as their communication channels. It is therefore mandatory, to reduce the communication overhead as much as possible, without making any compromises on the quality of information that has to be conveyed to the enterprise systems. Existing approaches optimizing the communication such as binary coding or compression of the data to be exchanged can be used.

If a different, more efficient communication format is used for communication within the front-end, this implies some format conversion at the interface to the back-end, residing at one ore more transition points. To prevent these transition points from being bottlenecks that introduce delay into the message path, as few messages as possible should run through them. This means that the communication during process execution should be as local as possible within one domain, minimizing the number of messages that have to cross the system boundary. Therefore, a process optimization algorithm has to distribute business process intelligence to several layers, execute it where it is most economical, i.e. where communication volume and format conversion frequency are minimal when interacting with the enterprise systems.

To include large quantities of networked embedded systems, the identity of each node should not be visible at the level of service orchestration. In an example application, a population of sensor nodes could monitor the conditions in a room to ensure that these conditions stay within some tolerable limits. Individual nodes would have services installed that collect sensor values. In this setting, from a back-end perspective, it is not desirable that the individual sensor reading services of each node are used. Instead, another high-level service that ensures the compliance with the given set of rules should discover and use the individual instances of the sensor reading service and assess the complete situation. When the cross-system process accesses the compliance service to get the current status, it should not need to be aware which node will answer his request, since this service is provided collaboratively by the whole sensor network. However, this notion of *anonymous* requests is not supported by current executable process description standards like the Business Process Execution Language (BPEL [2]) which would have to be extended. The next part includes a suggestion on how the extension of BPEL would fulfil the requirement.

Table 1. Parts of BPEL Ready for Cross-System Business Processes

Part of BPEL	Functionality	Use in goods receipt example
service interactions	sending[1] and receiving[2] web service calls	• actions of service instances are called to query for the shock / temperature status • use discovery service to fork request to all nodes in the delivery
sequences and flows	sequential and parallel chain of operations	• ask proxy node for temperature and shock status simultaneously
control statements	conditions (if), loops (for, while) etc.	• if the answer is positive, call ERP service for successful goods receipt, otherwise call ERP service for refusing the order
variables	store return values or call parameters of service invocations	• for temporarily storing the answer from service calls to make them accessible to control statements
logical operations	logical processing of request-response operations	• for combining the results of the two calls concerning shock and temperature
fault handling	compensate non-replying services	• To compensate if communication to devices on pallet is impossible

[1]needs a complete instance reference

[2]any incoming message can create a new instance of the process; sending and receiving can be combined to create synchronous, RPC-like service invocations

3. Optimization of Cross-System Business Processes

In this part, we propose a practice for modelling business processes for networked embedded systems based on the embedded services they offer. From the specification of the model to its execution, the steps shown in Figure 3 will be taken. First, a business process expert will model the process. Since s/he does not know the characteristics of the networked embedded systems that provide the embedded services, the model is likely to be inefficient in terms of locality of execution. In a second step, the process will be optimized. This step will be described in detail in this part. After the optimization, zero or more sub-processes will be identified that can be autonomously executed without back-end interaction in the embedded front-end. This process will be deployed to a run-time system in the front-end and for each initial request to the process, a process instance will be created that will be executed efficiently by the hybrid system.

3.1. Existing IT Standards around Business Processes

Currently, BPEL is regarded as the most widely used standard for describing business process in an executable way. BPEL is an XML format that allows the

specification of the participating parties (web services) in a business process, the operation of the parties that are invoked, and the sequences, conditions, and loops in which this takes place. By using process variables and simple arithmetic, the processing of input and output parameters to service invocations are described. Finally, BPEL also contains constructs for handling errors that occurred during the execution of a business process.

Another standard for describing business processes is the Process Specification Language (PSL [1]), a standardized (ISO 18629) ontology primarily for exchanging information about manufacturing processes between different applications, but also applicable to more general ones processes. The approach differs from classical, instructive representations of business processes as a declarative approach, stating facts (axioms), rules, and relations between processes, process occurrences, time points, and objects in Knowledge Interchange Format (KIF). This allows for automatic reasoning on the process descriptions, making it possible to detect contradictions that represent flaws in process definitions. If modelled in detail, it should also be possible to execute processes described in PSL. In this paper, we do not consider PSL for the pursued goal, but consider it worth investigating its applicability in the reasoning-based approaches in the future.

3.2. Current functionality of BEPL and needs for extension

The BPEL language currently supports sequences, service interactions, control statements, variables, logical operations, and error handling (see Table 1). These building blocks will be explained using a business scenario from the domain of supplier relationship management (SRM).

In the example scenario, a palette of ordered hard disks arrives at the goods receipt gate of a warehouse (see Figure 2). The supplier has sent an advance shipping notification, listing the identities of all disks that were issued. The disks are sensitive to mechanical shock and temperature. The receiving party only accepts the shipping if less than 1 % of the items have been exposed to shock or temperature beyond an allowed range. Small, battery-powered, wireless devices have been attached to each device, monitoring physical conditions of the disks throughout the shipping process. At the gate, the services on the devices are queried if the agreement is fulfilled. Additionally, a "coherence service" is initialized with the list of serial numbers of the shipment at goods issue. It monitors if any disk was taken from the shipment and log the time when this happened. If the shipping was handled correctly, the Enterprise Resource Planning (ERP) system of the customer accepts the shipment, sends a confirmation of the reception to the supplier, and adds it to the inventory of the warehouse. Otherwise, the shipment is rejected and a negative notification is sent to the supplier. The back-end warehouse management and SRM systems should be able to contact any node for the shipping's status, which should autonomously start the query and reply on behalf of all nodes.

Most of the business process for receiving the hard disks in this example can be expressed with the language constructs of BPEL as seen in Table 1. However, the dynamic nature of the scenario renders the addressing part of BPEL almost useless. It is built for static scenarios with well-known partners that do not change on a minute-by-minute basis. To be able to engage with anonymous wireless nodes, connecting and disconnecting from fixed infrastructure without prior notification, discovery protocols have to be used. Once, a physical connection has been established, discovery protocols from the web service world, such as WS-Discovery [4], which has been chosen as the discovery mechanism included in DPWS [3], can be used to identify the services on the mobile nodes.

The discovery and addressing function should be provided by a software framework and should be encapsulated, for uniform access from within the BPEL description, in a service as well. It should be possible to address whole service groups, i.e. all nodes that host a service of a certain type. Additionally to supporting discovery by service type, complex criteria for service selection should be allowed, including dynamically changing attributes like location and service state. If e.g. the service state encodes whether a regulation was violated in a node, the discovery system could then select all nodes where this condition is true.

The discovery service could either

i) just return a list of endpoints that can be iterated within the BPEL description, or
ii) completely care for distribution and result delivery, acting like a proxy for a request

In the first case, the list of endpoints has to be handled by the BPEL process, possibly resulting in high memory usage and outdated information when the environment is dynamic. In the second case, the request to the discovery service not only includes the selection criteria but also the request to be forwarded to the set of nodes, instructions if and how to collect responses (number of / list of responses, aggregated response, etc.), and mode of delivery (e.g. reliability, timeout). It should also be considered to extend BPEL by allowing new kinds of partner links that contain the addressing options or extend the flow element accordingly.

The alternative to placing the advanced discovery functionality into a framework around the BPEL engine would be to do a generic discovery and explicitly specify the selection in BPEL by discovering all services, going through the list of service endpoints, retrieving the additional information, and iterating the list. Due to the

verbosity of BPEL and the too high level of detail of service selection, this approach is discouraged.

3.3. Execution of the Business Process in the Front-end

When a BPEL process is to be executed in a hybrid system containing front-end and back-end, the part that runs in the front-end and does not need interaction with the back-end should be identified and detached as a sub-process. Preferably, the sub-process is also a BPEL process. If a centralized approach would be chosen (which is the default execution mode for BPEL today), where the state of the BPEL process is kept in an execution engine that resides in the back-end, there would be many service calls from this central entity into the front-end with immediate responses. In this centralized approach, the control flow continously passes between a central BPEL engine and services that are invoked.

To avoid this, the control should be passed from node to node among the embedded systems when no interaction with the back-end is necessary. This could be achieved e.g. by passing along tokens that contain a compact description of the relevant part of the business process along with the relevant execution state including a unique process ID. Those tokens have to be passed reliably to the next node, including e.g. re-sending in case of message loss. The process-IDs must be temporarily stored by nodes in order to drop identical incoming messages belonging to the same process instance. In the end, the distributed execution of the busieness process must produce the same result as a centralized exectuion would have produced.

In a *BPEL sequence* of invocations, only one node would be active at the same time. In case of a *BPEL flow* element, control would spread to the service instances that take part in the parallel action, and return to the service instance that is responsible for the next step after the flow.

The energy consumption during communication, especially in the wireless domain (e.g. wireless sensor networks), can be further improved by exploiting cross-layer optimization. To name just a few possiblities, strategies like adaptive periodic sleep, early radio shutdown, and delivery traffic indication could be used.

3.4. Optimisation of the Business Process Description

When a business process is modelled by a business process expert, s/he cannot be expected to know any details about the underlying services or their implementations. The resulting executable description might not be optimal in terms of minimal resource usage.

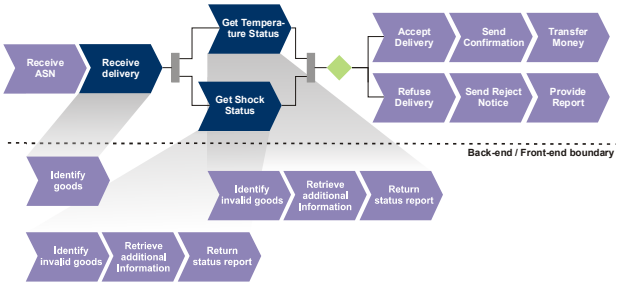

Figure 5. Example Business Process as modeled by business expert

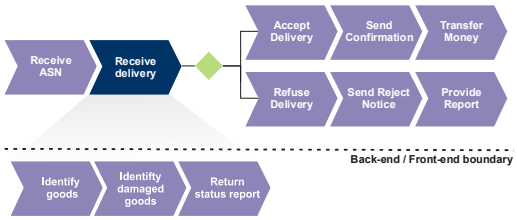

Figure 4. Example Business Process after optimization

One strategy, the separation of independent sub-processes, has already been outlined in this document. Applied recursively, this separation will lead to a tree of independent sub-processes. This means we are not only confronted with a split system of font-end and back-end, in the more general case there could also be three or more tiers, where each of the tiers has specific capabilities and restrictions.

Some services are tier-specific, meaning that they only exist in one tier, e.g. financial transaction service is likely to exist only in the back-end tier, while a pressure sensing service is more likely to be part of the front-end. In these cases, we have no freedom of choice regarding the location of the service to be used. Other services however, providing functions like aggregation, could be realized in any tier (possibly with varying functionality). They should be smartly placed or even replicated on multiple tiers in order to maximize locality of service interaction. Whenever there is either the choice to use a local service (i.e. one from the same tier) or to use a remote service (i.e. from another tier) that offer the same functionality, the local service should be preferred.

4. Example of a Device-Aware business process

In the example in Figure 5, a business process for receiving previously ordered goods (as described in 3.2, only the physical stress part) is depicted in a way it could be modelled from a business expert's perspective, where the process steps that make use of front-end services are marked in darker colour. Figure 4 depicts the same process as it is expected after it has undergone the proposed optimization.

In our example, when the supplier hands the delivery over to the carrier, it sends an *advanced shipping notification* that is received as the starting point of the business process, containing the identity of each item. When the delivery arrives at its destination, the business process is notified and it calls a front-end service that identifies the items in the delivery. The hard disks have to be checked for inappropriate delivery, i.e. if they had been exposed to unacceptable values of temperature or mechanical shock. Those tasks are carried out in parallel. Each of the two parallel process steps calls two front-end services. First, it queries for the identities of those items, where physical parameter limits have been exceeded. Then additional information is requested from each of the affected nodes (e.g. start and end times and maximum values of the incidents). The gathered information is then used for the decision whether the delivery should be accepted or rejected, depending on the rate of potentially damaged hard disks. In case of acceptance, a confirmation is sent to the supplier's ERP system and a money transfer is initiated. If the delivery is rejected, the supplier will be notified and a reason for rejection is given.

The given example process is likely to generate many messages that cross the back-end front-end boundary and can be reduced in the number of messages required to achieve the business goal. Using the automatic optimization described in Part 3, we expect to generate the more efficient process depicted in Figure 4. Here, the whole sub-process that leads to a decision whether to accept the delivery or not, is carried out by the embedded front-end and the communication crossing the system boundary is limited to a very simple request and a small result message.

5. Conclusion and Outlook

Networked embedded systems, in a corporate setting, deliver the highest return-on-investment (ROI) if they effectively support business processes. In this paper, we defined the concepts of back-end and front-end in hybrid systems of business software and networked embedded systems and showed ways of how these can be integrated. We used BPEL as an industry standard for designing optimized executable business processes.

The required functionality, to allow resource-constrained front-end devices participate in the execution of a business process, leads to an important postulation: the front-end's functionality has to be offered as services, either by direct service implementation on the devices or using a gateway that captures their functionality. This allows a business process expert to design processes that reach deeply into the front-end, making it more aware of the real-world situation rather than solely relying on (possibly outdated, erroneous, or incomplete) business data stored in databases. This implies not only real-world data flows to the business processes so that they can optimize their execution, but also the capability to delegate functionality (complete sub-processes) to the devices, therefore moving part of the business process intelligence and execution to other layers e.g. the device level. This leads to more agile enterprise systems and business processes that can better react to dynamic changes. Existing efforts towards zero-time enterprise modelling [7] can be further refined and the resulting models can be more efficient by taking into account the real-time dynamic specifics of the device infrastructure, something that up to now was hardly considered.

Finally, the requirements of a yet to formalized optimization algorithm are presented. The algorithm will restructures an executable business process description, as it is modelled by a business expert, for optimal execution within the hybrid system. While not changing the semantics of the process, the optimized version must take into account the varying capabilities of different classes of devices offering the services and minimizes energy-consuming communication within the front-end.

References

[1] C. Schlenoff, G. Tissot, J. Valois, J. Lubell, J. Lee, The Process Specification Language (PSL) 1.0, National Institute of Standards and Technology, 2000.

[2] T. Andrews, et al., Business Process Execution Language for Web Services Version (BPEL4WS) 1.1, IBM, BEA, Microsoft, SAP, Siebel, 2003, http://www.ibm.com/developerworks/library/specification/ws-bpel/

[3] Shannon Chan et al., Devices Profile for Web Services Specifications, February 2006, http://schemas.xmlsoap.org/ws/2006/02/devprof/

[4] John Beatty et al., Web Services Dynamic Discovery (WS-Discovery) Specifications, April 2005, http://schemas.xmlsoap.org/ws/2005/04/discovery/

[5] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S. and Winer, D., 2000. Simple Object Access Protocol (SOAP) 1.1. (http://www.w3.org/TR/SOAP/).

[6] Fleisch, E., Mattern, F. 2005. *Das Internet der Dinge: Ubiquitous Computing Und RFID in Der Praxis: Visionen, Technologien, Anwendungen, Handlungsanleitungen.* June 2005, Springer, Berlin, ISBN: 3540240039.

[7] Tan, W., Li, S., and Yang, F. 2005. Zero-Time Enterprise Modeling with Component Assembly and Process Model Optimization Techniques, *In Proceedings of the Fifth international Conference on Computer and information Technology (September 21 - 23, 2005). CIT. IEEE Computer Society, Washington, DC, 1135-1139.*

[8] Web Services Standards Overview, innoQ, Q1 2007, http://www.innoq.com/soa/ws-standards/poster/

[9] Saif, U., Greaves, Communication Primitives for Ubiquitous Systems or RPC Considered Harmful, *Proceedings of ICDCS International Workshop on Smart Appliances and Wearable Computing, IEEE Computer Society, 2001*