A GSM-based approach for Monitoring Cross-Organization Business Processes using Smart Objects

Luciano Baresi[1], Giovanni Meroni[1], and Pierluigi Plebani[1]

Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
[firstname].[lastname]@polimi.it

Summary. The execution of cross-organization business processes often implies the exchange of physical goods without necessarily changing the ownership of such goods. Typical examples are logistic processes where goods are managed by shipping companies which are not the owner of the goods. To ensure that these goods are properly handled while the service is executed, a monitoring system needs to be put in place.
The goal of this paper is to propose a novel approach for monitoring physical goods while executing cross-organization business processes. The approach envisions the usage of Smart Objects attached to the physical goods. To this aim, an extension of the Guard-Stage-Milestone framework is proposed to enable the Smart Objects to monitor the process execution also considering the limitations of their power and computational resources.

Key words: Guard-Stage-Milestone framework, Process-aware Smart Object, Process monitoring

1 Introduction

The exchange of goods in cross-organization business processes does not necessarily imply a change of ownership. Organizations could send goods to other organizations through a service provided by other organizations. A typical example is an order-to-delivery process: a seller company, which wants to send goods to the final customers, leaves their goods under the responsibility of a shipping company without transferring the property of these goods. At the same time, the owner of the goods (that could be either the seller or the receiver) wants to know how the goods are managed when they are under the responsibility of the shipping company. To this aim, Service Level Agreements (SLAs) are established among the parties. Although SLAs are commonly used in cross-organization business processes, the following limitations apply:

– Time consumption: the definition of SLAs is a long process where the quality of services, terms, conditions, and responsibilities need to be defined.
– Lack of flexibility: once defined, the SLA is valid for a specific service provider. Changing the provider usually requires the definition of a new SLA. Thus, this

approach is not suitable for highly dynamic environments where parties change frequently.
- Information hiding: when resources are under the control of the service provider, the status of the goods are monitored by the provider according to its capabilities.
- Activity status hiding: the owners of the goods have limited view on what happens during the execution of the process by the service provider.

The goal of this paper is to investigate how the adoption of Smart Objects can improve the monitoring of business processes and solve the aforementioned limitations. The main idea is to equip the goods that move among the parties with Smart Objects to monitor the status of the goods independently of the service provider that is in charge of them. To this aim, we propose a framework based on an extension of the Guard-Stage-Milestone (GSM) notation [2] to model the external activities that need to be monitored. As the monitored processes can last for long periods, the proposed approach takes care of the limitations of Smart Objects (especially in terms of battery life) to maintain a sufficient quality of the monitoring activity.

The proposed solution enables the design and execution of more flexible multi-party business processes. The monitoring logic is executed on a Smart Object attached to monitored goods as defined by the party that owns the resource. When the resource is under the control of another party in the business process, only infrastructural capabilities can be required to that party, as the monitoring is exclusively under the control of the Smart Object. The resource owner can check the status of goods during the whole lifecycle by simply connecting - if a connection is available - to the Smart Object itself. Moreover, the Smart Object is able to detect anomalous situations that violate the agreement and to promptly report the error to the owner.

The rest of this document is structured as follows. Section 2 motivates the paper by relying on a running example about multi-modal transportation. Section 3 highlights the limitations of traditional process modeling notations and proposes a new one one conceived specifically for Smart Objects. Section 4 proposes our solution to support the monitoring of multi-party business processes. Section 5 shows a possible application of our solution to the aforementioned example. Section 6 analyzes the state of the art, and Section 7 concludes the paper.

2 Processes and Smart Objects

Cross-organization business processes model the dependencies among processes performed by different organizations. An example is reported on the left-hand side of Figure 1: it refers to a manufacturing company that, after completing the realization of a product, sends it to the customer. To achieve this objective, the company relies on a shipping company that enacts the related process once contacted by the manufacturing company.

Once the goods are under its responsibility, the shipping company has to monitor how the goods are managed and report to the client if critical situations occur. What to monitor, and how the communication among the parties occurs, is properly ruled by a SLA defined in advance. As mentioned in the previous section, usually the service provider (i.e., the shipping company) is in charge of monitoring the service execution. From a business process perspective, once the goods are under the responsibility of the organization that provides the service, the same organization has to put in place all the tools able to monitor the service provisioning to comply with the SLA. As a consequence, each service provider always gives the same structure and type of monitoring information to all its service consumers, regardless of their specific needs. For instance, the shipping company can inform the final customer that the goods are about to arrive only when the truck leaves the warehouse. For some of the clients of the shipping company, this information is too coarse-grained and a notification to the final customer is desirable when the truck is in the same city as the final customer.

Another issue concerns the visibility of what happens during service provisioning. Referring to our example, if the shipping process fails, the manufacturing company could not necessarily be aware of when exactly the problem occurred. Moreover, it might happen that the shipping company notifies problems with some delay, and this could affect the recovery mechanisms that the manufacturer company could put in place.

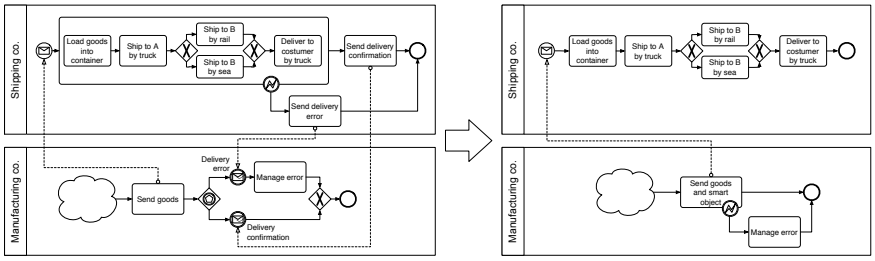

Fig. 1. As-is (left) and To-be (right) processes that adopt Smart Objects.

The approach proposed in this paper envisions the use of Smart Objects with the goal of making the relationship among the parties in cross-organization business processes simpler and more flexible.

A Smart Object is defined as "*an autonomous, physical digital object augmented with sensing/actuating, processing, storing, and networking capabilities*" [1]. Because of their diffusion, even in the domain of business processes, solutions based on Smart Objects are becoming more and more common.

In our approach, we assume that goods that are moving from different organizations are equipped with Smart Objects composed of: a computational unit able to run software, a sensing system, and a communication interface.

The use of Smart Objects permits to move from the situation reported on the left-hand side of Figure 1 to what is represented on the right-hand side. The

manufacturing company sends the goods along with the Smart Objects that have been properly configured to track the location of the goods with the required granularity. By doing so, if the activities composing the shipping company's process are publicly described, the Smart Object will also detect the phase of such a process that is currently being executed. This way, and assuming that the Smart Object can communicate via a broadband network, the manufacturer company can monitor where the goods are, in which status, and in which activity they are involved. The Smart Object can also identify violations in the process model and notify them to the user as soon as they happen (provided that a connection is available). By doing so, possible critical situations can be detected and managed directly by the owner of the goods, without asking the other organization to set up a specific monitoring infrastructure and to notify errors during the execution. Note that we may have the problem of having the Smart Object back once the shipping concludes. However, we decide not to address this problem in this paper, as it is similar to traditional ones like the management of pallets or shipping containers in the transportation domain.

The investigation on the adoption of Smart Objects for monitoring goods has been conducted in this paper by also considering the limitations of this technology. First of all energy consumption: most of these devices are battery-powered with limited autonomy and the battery is often difficult to recharge or replace. Fortunately, the amount of energy required by the computational part has dropped significantly in the last years, thus allowing Smart Objects to last longer and run more sophisticated software. For wireless data transmission, on the other hand, the reduction of energy requirements has not been as pronounced. For this reason, the proposed approach aim to increase the battery life by reducing the communications to the bare minimum. Secondly, the limitation of available computational resources on Smart Objects also influence the proposed solution. Monitoring the correct execution of the business process directly on board is not possible due to the requirements that usually characterize a business process management system. However, this paper envisions Smart Objects whose computational power is equivalent to the one of current Single Board Computing devices, such as Intel Galileo [**?**] or BeagleBoard [**?**].

Pairing a Smart Object with the goods to be monitored could have significant impact on costs. A cost-benefit model is discussed in [13]. Here the authors propose a model to estimate the impact of introducing Smart Objects to monitor the supply chain with respect to the sustained adoption costs and the gain in productivity. Even though we have not addressed cost related problems yet in our work, we plan to do so in the future and we will use such a model as a starting point.

3 Extended Guard-Stage-Milestone

According to our approach, the main task of the Smart Object is to check if the monitored goods are managed according to the process agreed among the organizations. Although control-flow modeling languages are suitable for defining

such an agreement (see Figure 1), the same representation cannot be used to instruct the Smart Object about the process to be monitored for several reasons.

First of all, if the Smart Object is fed with a control-flow description and an activity is not executed in the right order, the Smart Object raises an exception and the rest of the process could be not monitored. On the contrary, monitoring as to keep going, as the Smart Object could not always be connected and it could report anomalies in the process execution only at the end of the process execution.

Secondly, control-flow languages assume the presence of an orchestrator that explicitly starts the execution of the activities. In our case, the Smart Object has to autonomously realize when activities start and terminate as a direct and continuous connection with orchestrator could not exist.

Finally, control-flow languages lack constructs for explicitly defining conditions that detect the incorrect execution of an activity without necessarily implying a termination of such an activity. For example, during the shipping of a fragile item, the package could be dropped, condition that could cause a damage that would invalidate the whole process. Knowing exactly which activities were not correctly executed is critical for identifying responsibilities, and may also be useful to drive process changes.

Declarative languages, on the other hand, are well suited to our scenario. In fact, rather than relying on an explicit control flow definition, they mainly focus on defining which tasks should be performed under certain conditions, thus offering more flexibility with respect to control-flow languages. For this reason, the Guard-Stage-Milestone (GSM) [2] declarative notation was adopted and extended to properly instrument Smart Objects. We chose GSM because, with respect to other declarative languages, such as Declare [**?**], it provides constructs, namely *Guards* and *Milestones*, that explicitly define when an activity, which is named *Stage*, should start or end. Boolean formulas, named sentries, are associated with *Guards* and *Milestones*. They define conditions on captured events and, when they become true, determine the activation of the associated construct.

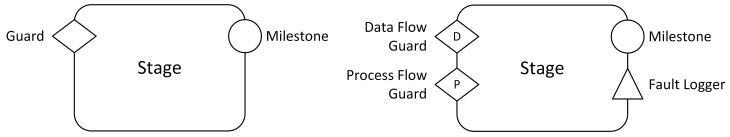

Fig. 2. Extended GSM (right) versus standard GSM (left). The Information Model element, which is part of the standard GSM specifications, is not depicted as our extension does not make changes to that part.

With respect to the standard definition of GSM, we introduce the following changes, as shown in Figure 2:

– Guards are divided into *Process Flow Guards* and *Data Flow Guards*: Process Flow Guards define sentries only related to the activation of Data Flow Guards

or Milestones, whereas Data Flow Guards define sentries only on external events. Defining conditions on both Milestones and data artifacts into the same sentry is not allowed. Moreover, Process Flow Guards do not determine the activation of the associated Stage. Instead, they specify which Stages are expected to start or end before the associated one.
For example, we know that an activity is executed whenever the Smart Object reaches a precise location. The Smart Object could be aware of the beginning of such an activity thank to the associated Data Flow Guard. In fact, we can define a sentry that is triggered whenever the Smart Object's GPS coordinates change and are equal to the ones of such a location.

– For each Stage, it is possible to define *Fault Loggers*: such annotations allow us to define a sentry which, if true, marks the associated Stage as faulty. However, despite Milestones, the associated Stage is not closed once Fault Loggers are triggered. Analogously, Fault Loggers differentiate from invalidating sentries since they do not cause completed Stages to be started again, and are ignored once the associated Stage terminates.
For example, during the execution of an activity, we want to make sure that the Smart Object's temperature will not exceed 50°C. By defining for such an activity a Fault Logger with a sentry that is triggered when the temperature changes and is above 50°C, we ensure that the activity execution will be successful as such Fault Logger is not triggered.

With these extensions GSM allows us to easily model process specifications suited for driving process-aware Smart Objects. Process Flow Guards model the process flow. Based on data gathered by the sensors installed on Smart Objects, Data Flow Guards define activity start conditions, Milestones activity end conditions, and Fault Loggers model activity constraints.

4 Process-aware Smart Objects

After introducing our extended GSM for Smart Objects, we can now present the software architecture deployed on Smart Objects which allows them to monitor processes. Figure 3 shows its main software components and their relationships.

– *Trace Generator*: is responsible for analyzing sensor data and external messages received by the Smart Object. It mainly consists of a Complex Event Processing (CEP) engine, which compares these data streams with the sentries defined in the process model to detect process events. The output of this module is a process trace, which records chronological information concerning the Data Flow Guards, Milestones or Fault Loggers that are triggered by a specific event.
– *Violations Detector*: compares the process trace with the process model to detect control flow violations and activity faults. When a connection is available, it informs the Smart Object's owner about violations by sending notifications. A notification cache is also adopted to temporarily store unsent notifications.

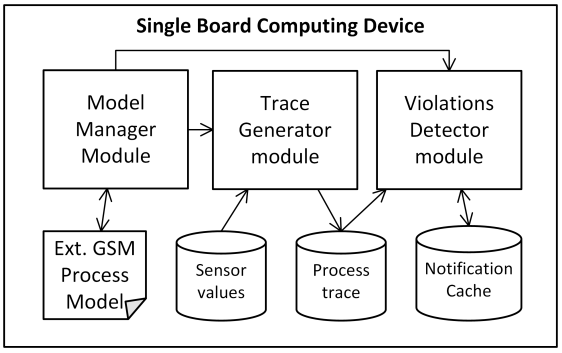

Fig. 3. Smart Object software architecture.

– *Model Manager*: is responsible for keeping the process model up to date. It accomplishes such a task by both receiving explicit updates by the process owners and requesting the process definitions to the involved parties that have not provided such definitions yet. It is also responsible for notifying changes on the process model to the other modules.

During process execution, changes in activity states are determined by Data Flow Guards and Milestones. Process Flow Guards and Fault Loggers, on the other hand, are used to determine violations on process flow and activity data, respectively.

Process-aware Smart Objects operate according to a GSM-based process model, while the agreement among organizations is usually defined by using a control-flow based model. The following steps must be performed to facilitate the definition of the GSM-based process model that is then deployed on the Smart Objects (see Figure 4) [1]. The starting points are the control-flow based process and the to-be-monitored goods.

– *Identification of activities.* Starting from a standard BPMN process definition, the user identifies the activities that wants to monitor by the Smart Objects. By doing so, a view on the main process is created and used as a reference for the subsequent phases. Such view contains only the portion of the process definition related to the to-be-monitored activities.
– *Generation of extended GSM definition.* Starting from the process view of the previous phase, the system semi-automatically generates the process definition by using the extended GSM notation. This is made possible by automatically transforming activities in Stages, activity flows in sentries for the Process Flow Guards, and gateway conditions in sentries for the Data Flow Guards. An empty Milestone is also inserted for each activity to allow for the definition of sentries in Process Flow Guards. The translation rules for the message

[1] For the sake of clarity, we assume that BPMN is adopted as control-flow process modeling language. Anyway, other control-flow languages can be adopted without affecting the validity of the approach.

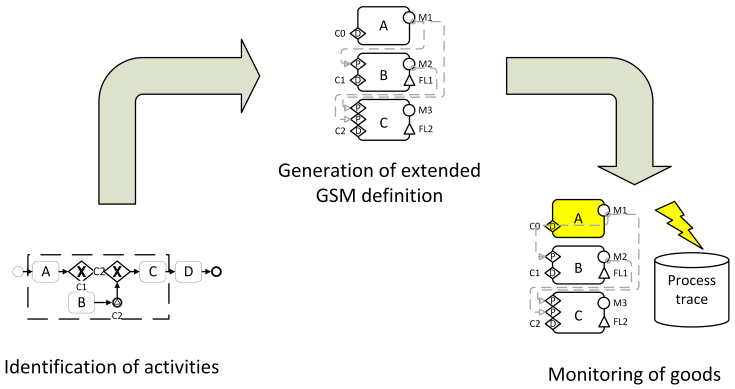

Fig. 4. GSM-based Passive Smart Objects integration framework.

flow and process events are currently under investigation. Starting from the automatically generated model, the user manually enriches it by specifying sentries for empty milestones and optionally adding additional Data Flow Guards and Fault Loggers. The resulting GSM schema will be deployed on the Smart Objects.

– *Monitoring of goods.* In this phase the process definition is loaded onto the Smart Objects and executed. During execution, the Smart Objects keeps track of the actual process trace: the actual Stage start and termination order is recorded, together with the list of Stages whose Fault Logger conditions are triggered, and those whose Data Flow Guards are triggered before Process Flow Guards become valid. With this information, process compliance can be assessed at runtime and violations can be promptly reported.

5 Validation

Considering the cross-organization process model introduced in Section 2, the manufacturing company wants to monitor the activities performed by the shipping company while the goods are moved to the final destination. As shown in Figure 5, the shipping process to be monitored is composed of the following steps: (i) the goods are stored into a shipping container attached to a truck; (ii) the truck ships the goods to site A, more precisely to the railway station if the shipping takes place during holiday, or to the seaport if it takes place during a working day; (iii) the container is detached from the truck and either loaded onto a train, which carries it up to site B, or loaded onto a ship bound for site C; (iv) the container is unloaded from the train or the ship, and attached to a truck that finally ships the goods to the customer. This process has been simplified on purpose with respect to the real world scenario to easily understand its transformation into extended GSM. That said, future work will take into consideration more complex process models.

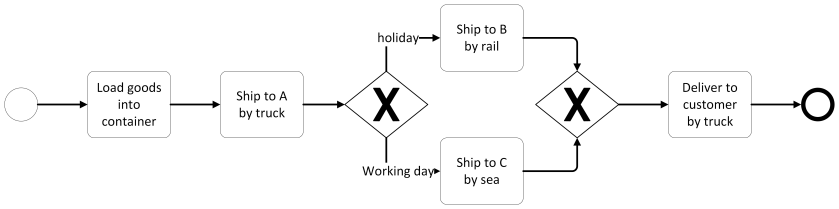

Fig. 5. Shipping process in BPMN notation.

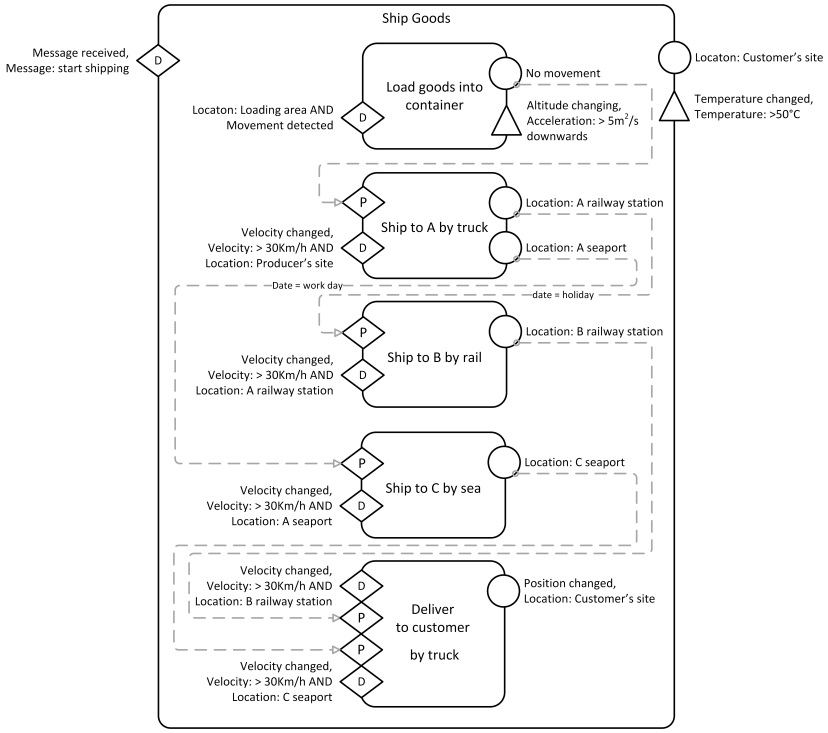

Fig. 6. Shipping process in extended GSM notation.

From this process, the GSM-based process definition reported in Figure 6 is obtained. Every stage has a Data Flow Guard that defines the conditions on the data that determine the start of each activity. In this case, the Smart Object can identify the beginning of the goods loading phase by detecting that it is being moved and its GPS coordinates identify the loading area of the producer's site as the current location.

Also, every Stage has a Milestone that defines the condition on data that determines the end of each phase. In this case, the Smart Object can identify the end of the rail shipping phase by detecting that its position has changed and its GPS coordinates identify site B railway station as current location.

The definition of Process Flow Guards allows us to define the expected activity execution order, which permits to assess process compliance with respect to the process flow. In this case, by defining for stage *Deliver to customer by truck* two Process Flow Guards connected to the milestone of stage *Ship to B by rail* and *Ship to C by sea* respectively, we expect either the rail shipping to site B or the sea shipping to site C to precede the final delivery of the goods to the customer.

Fault Loggers, conditions that, if true, mark the associated activity as invalid, are also defined. In this case, on Stage *Load goods into container* we define a Fault Logger on the maximum vertical acceleration of the Smart Object. This enables to detect if the goods have been dropped and thus it identifies that the associated phase was not carried out correctly.

The obtained extended GSM schema is deployed on a Smart Object equipped with sensors monitoring the conditions that affect the shipping process (such as temperature, position and altitude). Indeed, these are the sensors required to realize if activities are skipped (e.g., the goods are shipped neither by rail nor by sea), executed in the wrong order, or not correctly performed (e.g., the goods were dropped during the loading operation).

The Smart Object can generate process traces during the execution of the shipping process. This way, both the producer and the shipping company can be notified whenever fault conditions are triggered or violations in the activity execution order are detected. Moreover, the shipping company can collect process traces from each Smart Object, and change its original BPMN process definition to reflect how the process is actually executed.

6 Related work

Some research efforts have been spent on integrating Smart Objects with business processes. Meyer et al. [3] propose to extend the BPMN 2.0 notation to model smart devices as process components. This approach keeps the process knowledge on the information system, and no process fragments are introduced on smart devices.

Thoma et al. [4] propose to model the interaction with Smart Objects in BPMN 2.0 as activity invocations for simple objects, or as message exchanges with pools representing the whole Smart Object for more complex ones, thus leaving space for distributing part of the process definitions on Smart Objects. The limitation of this work is the lack of details concerning how to deal with data uncertainty and how to define data requirements.

Tranquillini et al. [5] propose a framework that employs BPMN for driving the configuration of a Wireless Sensor Network (WSN). Since BPMN is used only at design time for defining the business process, and then it is converted into binary code executable by the WSN, introducing changes in the process definition at runtime is difficult. Also, simultaneously supporting multiple processes on the WSN is not feasible with this framework.

Schief et al. [6] propose a centralized framework that extends the process design and execution phases of BPM and that takes into consideration events generated by Smart Objects. Furthermore, such a framework provides data quality mechanisms for evaluating events and sensor data. However, this framework does not allow the explicit definition of requirements for sensor data.

In particular, concerning goods tracking and monitoring among different parties, the solutions currently adopted by companies are limited to identifying when goods enter or exit a specific location by using passive RFID tags attached to them. For example, Brizzi et al. [7] propose a middleware that supports the exchange of RFID tag information across federated companies. However, in such solutions no process definition exists, therefore the task of identifying process violations is left to the information systems that belong to the involved parties.

Kunz et al. [8], on the other hand, propose a framework for monitoring goods that takes into consideration the involved business process. Such a framework, uses a CEP engine to collect and process sensor data coming from RFID tags attached to goods and to transform them into events. These events are then sent to a workflow engine that detects violations in the expected process flow and reacts by running compensation activities. However, such a solution requires the user to define process flow violation conditions at design time by specifying them with the BPMN Escalation event. Therefore, unpredicted violations cannot be handled during execution.

7 Conclusions and Future work

This paper presents an approach for monitoring physical goods when they are exchanged among different parties according to a defined cross-organization business process. As control-flow models are not suitable to run on the behavior of Smart Objects due to the limited resources, and are not suitable for defining the monitoring system behaviour due to their lack of flexibility, an extension of the Guard-Stage-Milestone framework is proposed. With this extension, a GSM-based definition of the portion of process to be monitored is deployed on Smart Objects. These objects, traveling along with the monitored physical goods, realize if and when anomalies in conducting the process occur.

Future work will mainly concentrate on a better definition of how to automatically derive the GSM process model that has to be deployed on the Smart Object starting from the control-flow process model agreed by the organizations. Moreover, an implementation of the modules composing the process-aware Smart Object is planned to validate the approach on a real testbed.

Acknowledgments

This work has been partially funded by the Italian Project ITS Italy 2020 under the Technological National Clusters program.

References

1. Fortino, G., Trunfio, P.: Internet of Things Based on Smart Objects. Technology, Middleware and Applications. Springer International Publishing (2014)
2. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, Fenno(Terry), I., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In Bravetti, M., Bultan, T., eds.: Web Services and Formal Methods. Volume 6551 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 1–24
3. Meyer, S., Ruppen, A., Magerkurth, C.: Internet of things-aware process modeling: Integrating iot devices as business process resources. In: CAISE 2013. LNCS 7908. Springer Berlin Heidelberg (2013) 84–98
4. Thoma, M., Meyer, S., Sperner, K., Meissner, S., Braun, T.: On iot-services: Survey, classification and enterprise integration. In: IEEE GreenCom 2012. (Nov 2012) 257–260
5. Tranquillini, S., Spieß, P., Daniel, F., Karnouskos, S., Casati, F., Oertel, N., Mottola, L., Oppermann, F., Picco, G., Römer, K., Voigt, T.: Process-based design and integration of wireless sensor network applications. In: Proc. BPM 2012, Berlin, Heidelberg, Springer-Verlag (2012) 134–149
6. Schief, M., Kuhn, C., Rsch, P., Stoitsev, T.: Enabling business process integration of iot-events to the benefit of sustainable logistics. Technical report, Darmstadt Technical University (2011)
7. Brizzi, P., Lotito, A., Ferrera, E., Conzon, D., Tomasi, R., Spirito, M.: Enhancing traceability and industrial process automation through the virtus middleware. In: Proceedings of the Middleware 2011 Industry Track Workshop, ACM (2011) 2
8. Kunz, S., Fabian, B., Ziekow, H., Bade, D.: From smart objects to smarter workflows–an architectural approach. In: Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International, IEEE (2011) 194–203
9. Kharbili, M.E., de Medeiros, A., Stein, S., van der Aalst, W.: Business process compliance checking: Current state and future challenges. In: Modellierung betrieblicher Informationssysteme - Modellierung zwischen SOA und Compliance Management. (Nov 2008) 107–113
10. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. In: Proc. of ICSOC-ServiceWave '09, Berlin, Heidelberg, Springer-Verlag (2009) 500–515
11. Ly, L., Rinderle-Ma, S., Gser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems. Information Systems Frontiers **14**(2) (2012) 195–219
12. Weidlich, M., Ziekow, H., Mendling, J., Gnther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Business Process Management. LNCS 6896. Springer Berlin Heidelberg (2011) 182–198
13. Decker, C., Berchtold, M., Chaves, L.W.F., Beigl, M., Roehr, D., Riedel, T., Beuster, M., Herzog, T., Herzig, D.: Cost-benefit model for smart items in the supply chain. In: The Internet of Things. Springer (2008) 155–172