Realizing Process Modifications in Container Terminals with SOA – A Prototype

Thomas Will and Thorsten Blecker

Hamburg University of Technology (TUHH),
Schwarzenbergstr. 95, 21073 Hamburg, Germany
publications@thomas-will.eu, blecker@ieee.org

Abstract. This paper deals with changes in existing IT systems resulting from RFID-based process modifications in maritime container logistics. The article demonstrates how a SOA enables this IT adoption by functioning as a software layer between process and legacy system to enable flexible processes. In addition, the paper shows how we applied the service principles during the design and the implementation phase and perform the implementation by developing a physical miniature prototype to visually show how the modifications can easily be implemented by service-based computing.

Keywords: Service-based Computing, Legacy Systems, RFID, Container Logistics, Process Modifications, Prototype.

1 Introduction

Container logistics is a growing industrial sector. Container terminals have been realizing average annual growth rates up to 11 percent since 1995. Current and expected volumes overstrain the existing infrastructure. Therefore, responsible operators are searching for new technologies to accelerate existing processes by reducing errors, manual intervention and redundancies. Radio Frequency Identification (RFID), lately standardized for container logistics by the International Organization for Standardization (ISO) [1], is such a technology. The ISO standardized three different transponders for maritime containers [2]: the license plate [3] [4], which stores container specific data, the shipment tag [5], which stores cargo / shipment specific data, and the electronic seal [6-11], which serves as a mechanical seal covering some additional features.

Introducing RFID to container logistics means integrating this technology into existing logistics processes. This paper illustrates how resulting changes in the IT infrastructure (while retaining existing IT systems) can be covered by service-based computing and how we employed service design principles during our SOA design.

To give an overview, the next chapter briefly explains the general process structure followed by the example of a transhipment process. Chapter three deals with the service design in this example and illustrates how service design principles have been applied in our scenario. Finally, chapter four provides details on the developed prototype and further activities.

E.M. Maximilien et al. (Eds): ICSOC 2010 Workshops, LNCS 6568, pp. 253–264, 2011.

2 Process Analysis and Modifications

RFID technology itself does not provide benefits unless "interacting" with the business process [12]. Thus, this section analyses the process steps a container takes while being transported from a depot to a shipper and via an intermodal transport chain to a consignee and back to another depot. It should be noted that each container transport is different, and this reference process can only illustrate a generic form of existing processes. The objective of the accomplished analysis is to detect the interdependences between material flow (container) and control flow (information) in order to provide a basis for the service design.

2.1 Research Methodology

The process analysis is based on expert consultations supported by *the Logistics-Initiative Hamburg* [13], as well as expert interviews with logistics service providers. The Logistics-Initiative Hamburg, founded by the Hamburg Business Development Corporation [14], is a registered association of companies and institutions working in the Hamburg metropolitan region. It aims to develop and expand the area of Hamburg as a logistics hub. The members decided to launch a working group to investigate the use of RFID in container logistics. This group consists of 20 experts working for logistics infrastructure providers (e.g. terminal operator), ocean carriers, logistics service providers, insurance companies and IT consultancies. In fact, there has been a core group of four experts (a terminal operator, a logistics service provider, an ocean carrier and an IT service provider) which provided the main part of relevant details for the analysis. The process analysis follows Becker's seven step waterfall model [15]: (1) define team objectives, (2) assemble team, (3) define process, (4) gather information, (5) record process, (6) document process and (7) validate process.

2.2 Process Structure

Transport, transhipment and inventory processes are the core elements of a transport chain. During the transport process, a container is loaded on a truck (T1), railway wagon (T2) or in ship tonnage (T3). During the process analysis, we focused on the transhipment processes to identify useful readout points and process modifications. The reference process (Fig. 1) summarizes the 14 investigated transhipment processes.

To give an example, the following sections explain sub-process *no. 09 – Deliver via oversea vessel* and deal with the process modifications. Sub-process no. 09 is chosen, (1) because of its simplicity, which makes it easily understandable and extendable, (2) its high automation degree, i.e. proven RFID-based enhancements in this process are more difficult than in other processes, and (3) its relevance, because unloading containers from oversea vessels is a major bottleneck in current transport chains.

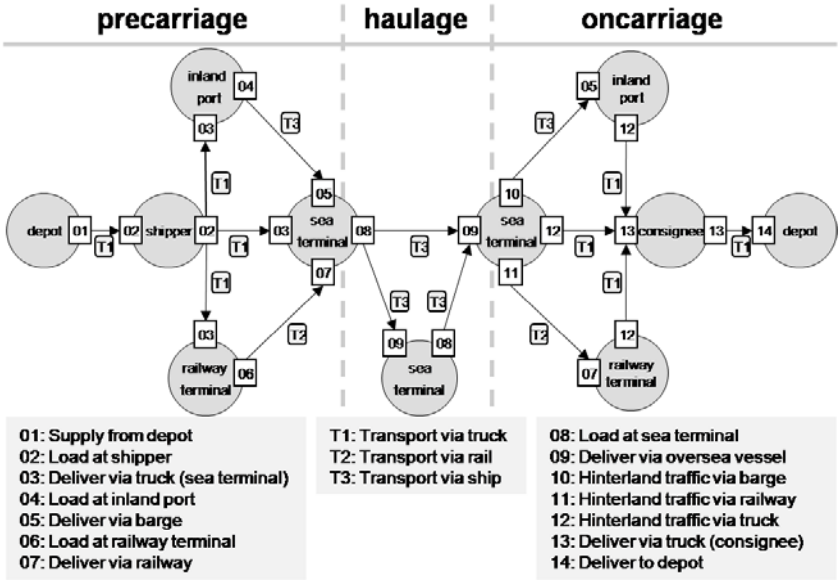

Fig. 1. Transhipment processes in intermodal container logistics

2.3 Analysis and Modifications – Sub-Process No. 09 – Deliver via Oversea Vessel

Phase 1 - data reception: As soon as the maritime carrier loads a container on his vessel, he informs the destination terminal on the container arrival. Just before the ship arrives, the maritime carrier submits the ship's stowage plan via BAPLIE[1] [16] [17], including the position of all loaded containers. Based on the advanced notification and the stowage plan, the sea terminal creates the unloading plan, which contains the sequence of transhipment processes to unload desired containers.

Phase 2 – data check: After the ship has docked, the unloading process starts. The gantry crane operator raises the container to be unloaded and another staff member checks the container number. If the container number matches with the number listed in the unloading plan, then this staff member sends the signal for dumping the container on the quay wall to the gantry crane operator. Other staff members remove the secure locks, check the container number once again and the availability of a seal, but not the seal number. Subsequent to the comparison, the data is stored in the system. In case of divergences during the checks, the container is handled manually.

Phase 3 – data forwarding: The container is stocked and the handling operator updates the unloading plan comprising the actual state of container unloads. Based on this, the handling operator creates a new stowage plan and submits it to the maritime carrier and to the next handling operator. The maritime carrier briefs the 3PL (third-party logistics provider) on the container arrival, which enables the 3PL to further

[1] BAPLIE means bay plan/stowage plan occupied and empty locations message.

specify the transport. The maritime carrier also accounts the container transport, and the 3PL creates transport orders for the oncarriage.

After developing the generic reference process, we identified the required modifications and those changes which allow enhancing the process when introducing RFID. Thus, the modifications are divided into three categories: (1) modifications that have to be accomplished to integrate RFID into the process, (2) those that accelerate the process, and (3) those that aim to extend the process functions without increasing lead time.[2] After providing an overview on the example process and the RFID-caused changes, the next chapter will explain the modifications and the resulting SOA design in more detail.

3 SOA Design

Introducing RFID to container logistics and integrating this technology into existing logistics processes leads to several process modifications; which results in IT infrastructure changes. This chapter demonstrates how a SOA facilitates this IT adoption by functioning as a software layer between process and legacy system to enable flexible processes.

Due to the fact that most of the different transhipment processes contain similar process steps that are currently implemented by a legacy application logic, the first step is to map the reusable legacy application logic into services to allow the reusability in the modified processes. Compounding these basic services that wrap the existing application logic and with additional services that provide simple and reusable utilities to the modified process, we create a basic service layer containing all legacy and newly developed basic applications. Following the naming of Thomas Erl [20], we label this layer as application service layer. Based heron, we develop services that compose available application services to execute the business logic required by the processes. These services are called value-added business services are located in the so called business service layer [20]. Finally, we developed a third service layer ontop of the existing layers that realized a workflow management by orchestrating the developed business services and enabling a mapping of the process logic via business service interaction. This topmost layer is called orchestration service layer [20].

We design the services regardless of the underlying IT system, because IT systems differ from terminal to terminal, i.e. the designed SOA is able to cope with all process steps, including the modifications without further support from an existing terminal IT system. Thus, depending on the terminal and its IT system, some of the developed services are not necessary, and can be replaced by services using the legacy system's function. The following sub-chapters first explain only those (13 out of 35) services that are used in sub-process no. 09 and second deal with the application of service design principles.

[2] Due to space restrictions, we skip the illustration of the "sub-process no. 09 – deliver via oversea vessel (process)". For an illustration of the sub processes, please consult [18]. For more detailed information on the accomplished process analysis and modification, please consult [19].

3.1 Designing Services– Sub-Process No. 09

The process-service connection are visualized with dashed rectangles which represent the newly designed services (Fig. 2). Sub-process no. 09 starts when the maritime carrier submits information about the arriving container to the sea terminal. The information consists of the container's registration and the stowage plan. For documentation purposes and because of the different submission times, the handling operator has to store both. Thus, we add two services *store container registration* and

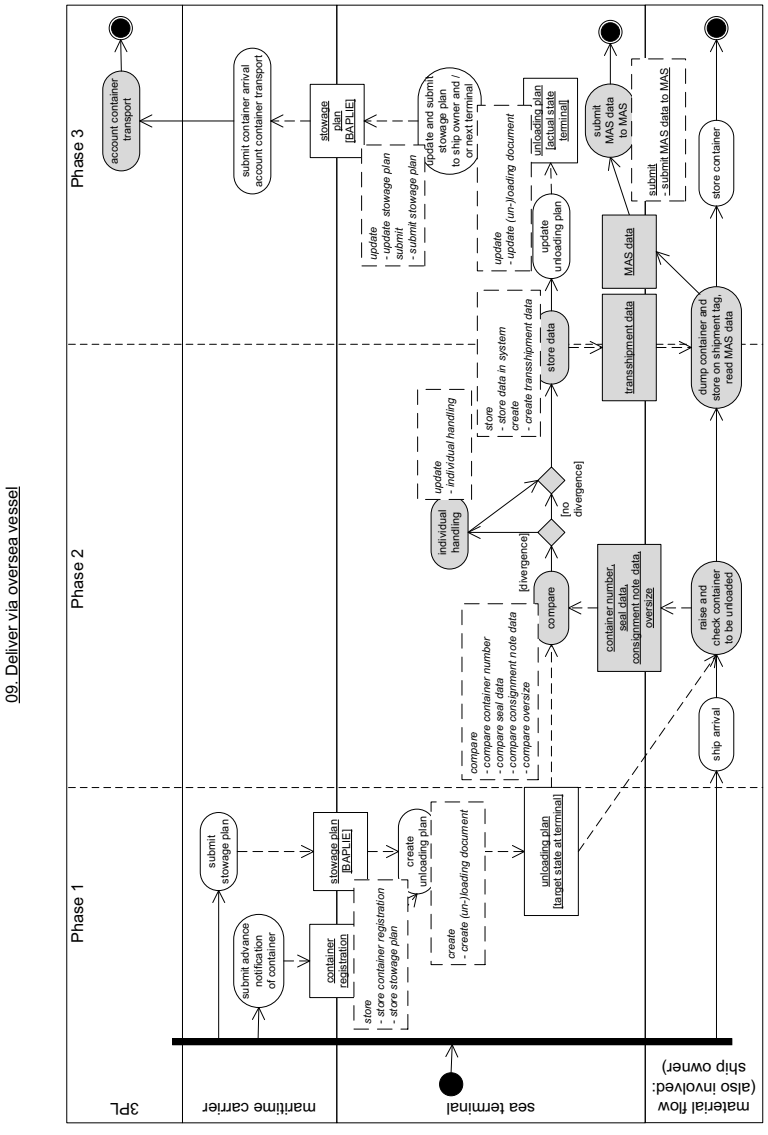

Fig. 2. Sub-process no. 09 – deliver via oversee vessel (services)

store consignment note data on the application service layer, which are compounded by the business service *store*.

After storing received information, the sea terminal has to create an unloading plan, which is realized by the application service *create (un-)loading document*. This service is covered by the business service *create*.

Today, the data comparison is a manual (paper-based) task. Using RFID requires the implementation of appropriate application services to realize above described modifications. The first process step in phase 2 contains several comparisons that are mapped to the corresponding service (in brackets): container number (*compare container number*), seal data (*compare seal data*), consignment note data (*compare consignment note data*) and oversize (*compare oversize*). A business service named *compare* comprises these application services. A divergence between the data stored in the IT system and the data stored on the transponder leads to individual handling (*individual handling*). This service provides an interface for human intervention and updates the system data, based on the human input. This application service is part of the business service *update*.

The system further stores the data (*store data in system*), creates the transhipment data and stores it on the transponder (*create transhipment data* and *store data on shipment tag*). Both application services are part of the business service *store*. After initiating the transhipment, the unloading plan and subsequently the stowage plan are updated (*update (un-)loading document* and *update stowage plan).* Finally, the terminal system submits the stowage plan to the maritime carrier (*submit stowage plan* – part of business service *submit*). Table 1 gives a brief overview on the services used in sub-process no. 09.

Table 1. Description of services used in sub-process no. 09

Service Name	Input (Source)	Output	Description
store container registration	container registration (3PL or maritime carrier)	finish /error notification	store input in the SOA
store stowage plan	stowage plan (ship forwarding agent or maritime carrier)	finish /error notification	store input in the SOA
store data in system	transshipment data (SOA) [3]	finish /error notification	store input in the SOA
store data on shipment tag	various inputs (SOA)[4]	finish /error notification	store input on shipment tag
compare container number	(un-)loading document (SOA) container number (RFID reader)	(non-)conformance notification	compare inputs
compare seal data	(un-)loading document (SOA) seal data (RFID reader)	(non-)conformance notification	compare inputs

[3] The here designed terminal SOA provides the data either by using functions from the existing legacy system (such as data storage) or by generating the data based on own computations.

[4] Depending on the process, the input is a combination of: consignment note (former paper-based), container number, eSeal number, customs declaration number (former paper-based), transhipment data (data of participants that were in contact with the container), truck transport information.

Table 1. *(continued)*

compare oversize	(un-)loading document (SOA) oversizes (RFID reader)	(non-)conformance notification	compare inputs
create (un-) loading document	various inputs (SOA) [5]	(un-)loading document[6]	match the stored input data to an (un-) loading plan and store it in the system
create transshipment data	(un-)loading document environment data[7]	transshipment data	create a data set that includes all necessary data of the transshipment.
update (un-)loading document	(un-)loading document (SOA) data of transhipped containers (RFID reader)	(un-)loading document	update (un-)loading document (based on an existing (un-)loading document) and data of transhipped containers
update stowage plan	stowage plan (SOA) (un-)loading document (SOA)	stowage plan	create a new stowage plan by updating the initial stowage plan with the accomplished (un-)loading document
individual handling	(un-)loading document (SOA) error notification (SOA) non-conformance notification (SOA)	(un-)loading document	solve the error / non-conformance by manual intervention
submit stowage plan	stowage plan (SOA)	finish /error notification	submit stowage plan to ship forwarding agent / maritime carrier

3.2 Applying Service Design Principles

After describing the developed application and business services for sub-process no. 09 in detail, we demonstrate how we applied design principles for the whole SOA to be able to accomplish the asked duties. Services have to be reusable, loosely coupled, composable, autonomous, stateless, discoverable, location transparent, share a formal contract, abstract the underlying logic, and have a network-addressable interface.

[5] The application service *create (un-)loading document* uses all data that was stored in previous process steps of this sub-process. Depending on the process, the input is a combination of: release order, container registration, customs clearance, consignment note data, sequence of wagons, stowage plan, registration stop, damaging, or truck license plate number.

[6] The service creates an unloading document if it is implemented in sub-process 3, 5, 7, 9, or 11 and a loading document if it is implemented in sub-process 4, 6, 8, 10, or 12. For sub-process 3 and 12, the list contains only one container. In these sub-processes, the (un-)loading plan is called shipment data.

[7] Environment data can include timestamp of the transhipment, identifier of the terminal and possibly data of the personnel who transhipped the container and the company that delivered the container to the terminal.

Services are reusable. Regardless of whether immediate reuse needs exist, services are designed to be potentially reusable. To fulfill this requirement services are designed generic, i.e. processing-specific logic is stored in exchanging messages. The service computes the result based on the given input and can therefore provide a general capability which can easily be reused. Table 2 focuses on the reuse of developed application services in the modified sub-processes 03 to 12 and illustrates the high reuse-rate of designed services (# displays the amount of reuse per service).

Table 2. Reuse of services used in sub-process no. 03 to 12

store services\ sub-process no.	3	4	5	6	7	8	9	10	11	12	#
store release order								X	X	X	3
store container registration	X		X		X		X				4
store customs clearance		X		X		X		X	X	X	6
store consignment note data	X	X	X	X	X	X	X	X	X	X	12
store sequence of wagons				X	X				X		3
store stowage plan		X	X			X	X	X			5
store registration stop		X				X					2
store damaging	X	X		X	X	X		X	X	X	8
store truck license plate number	X									X	2
store data in system	X	X	X	X	X	X	X	X	X	X	12
store data on shipment tag	X	X	X	X	X	X	X	X	X	X	12
compare services\ sub-process no.	3	4	5	6	7	8	9	10	11	12	#
compare sequence of wagons				X	X				X		3
compare position on wagon number				X	X				X		3
compare container number	X	X	X	X	X	X	X	X	X	X	12
compare consignment note	X		X		X					X	4
compare customs clearance										X	1
compare seal data	X	X	X	X	X	X	X	X	X	X	12
compare oversize	X		X		X		X				4
compare dangerous cargo label (yes/no)	X	X		X	X	X		X	X	X	10
compare truck license plate number										X	1
create services\ sub-process no.	3	4	5	6	7	8	9	10	11	12	#
create (un-)loading document	X	X	X	X	X	X	X	X	X	X	12
create shunting tasks				X					X		2
create interchange	X										1
create receipt										X	1
create transshipment data	X	X	X	X	X	X	X	X	X	X	12
update services\ sub-process no.	3	4	5	6	7	8	9	10	11	12	#
update (un-)loading document	X	X	X	X	X	X	X	X	X	X	12
update sequence of wagons				X	X				X		3
update stowage plan		X	X			X	X	X			5
update missing data	X				X					X	3
update consignment note data	X									X	2
individual handling	XX	X	X	XX	X	X	X	X	XX	X	15
submit services\ sub-process no.	3	4	5	6	7	8	9	10	11	12	#
submit sequence of wagons				X	X				X		3
submit stowage plan		X	X			X	X	X			5
submit consignment note data	X										1
submit status information										X	1

Services are loosely coupled. We established a loosely coupled relationship between services, their underlying application logic, respectively. The services are connected via formal contracts to each other, enables a coupling as long as this specific orchestration of operations / services is required. The service reusability illustrated in Table 2 further indicates that the services are loosely coupled, because they are reused in diverse sub-processes.

Services share a formal contract. In order to interact with each other, the services share specific information such as the service result, input and output message as well as rules and characteristics of the service and its operations. We use the XML specification WSDL [21], an open standard for these service descriptions.

Services abstract underlying logic. The visible part of a service is what is exposed via the service's description and formal contract. As we use the service-oriented computing as an intermediate layer between legacy system and process, the underlying logic is completely invisible and irrelevant to service consumers and there is no restriction concerning the size, allocation, distribution or complexity of the underlying application logic.

Services are composable. This principle ensures that our services can reuse other services to accomplish their work and do not need to redundantly implement specific operations. The service itself can also be part of other service compositions, which use its function to accomplish the own goals.

Services are autonomous. The underlying logic that is governed by a service works within an explicit boundary. For execution, this logic is not dependent on other applications outside the service's autonomy area. This does not necessarily mean, that a service has exclusive ownership of the encapsulated logic, but only guarantees control over underlying logic at the time of execution. As the existing legacy system functions are triggered by external inputs, mainly coming from manual data entries, we simply replace these manual inputs with RFID-reads, forwarded by the SOA and a service, respectively. Thus we can assure that the service has control over the legacy function at the time of execution.

Services are stateless. Statelessness is a required design principle to assure loosely coupled services and promote reusability. Our services minimize the amount of state information they manage and the duration they hold it. In fact, all services are stateless and receive all relevant state information via the incoming trigger-messages.

Services are discoverable. Each service provides a piece of processing logic, which might potentially be reusable. To prohibit redundant creation of services and underlying logic, being discoverable is an important service design criteria. Each service provides descriptions (WSDL) to be discovered and understood by humans as well as service consumers who may be able to make use of the services' logic.

Services have a network-addressable interface. Service requestors must be able to invoke a service across the network. Each service can be executed via an internal interface as well as via HTTP and can return the results via both ways.

Services are location transparent. Service consumers do not have to access a service using its absolute network address. They dynamically discover the location by simply submitting their request to a registry that triggers the corresponding service. This feature allows services to move from one location to another without affecting the consumers.

4 Prototype and Further Work

After designing the required services, we currently develop a real-world prototype that implements them. The prototype's goal is to illustrate the applicability of service oriented computing in the area of maritime container logistics to give logistics practitioners an understanding of this technology. Thus, the prototype is divided in two parts: (1) the service oriented computing discussed in this paper and (2) a physical miniature prototype of a container terminal and transport vehicles (ship, train, truck) mapping the above presented processes.

Fig. 3 provides a schematic overview on the container terminal, while Fig. 4 shows the real prototype implementation.

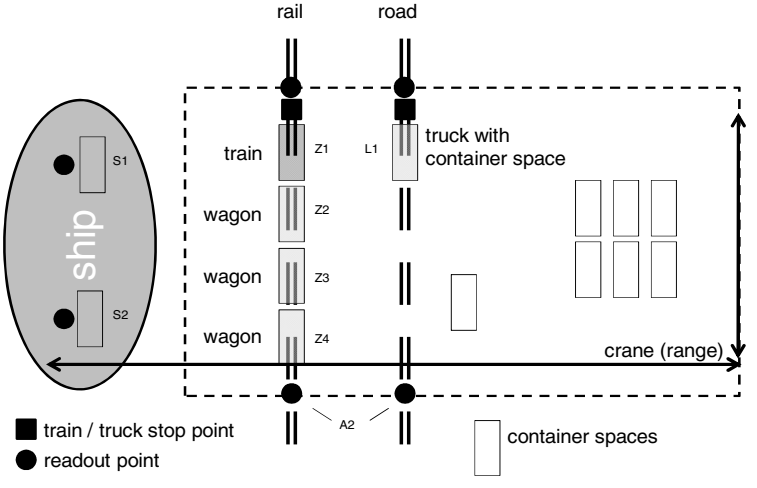

Fig. 3. Physical miniature prototype – schematic terminal view

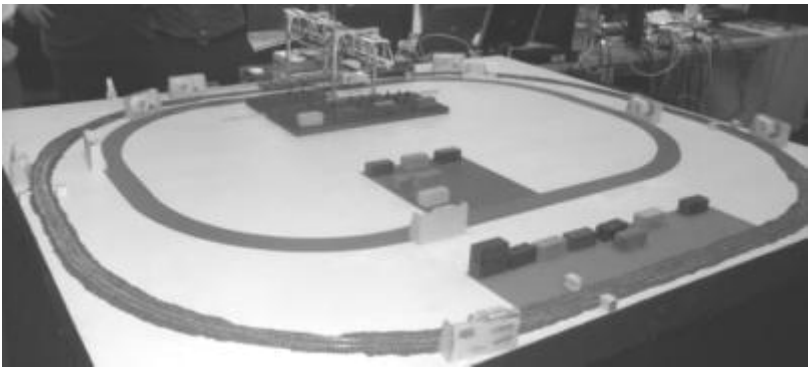

Fig. 4. Physical miniature prototype – real implementation view

Besides implementing the terminal-internal processes with the SOA, the prototype should also focus on cross-company processes. Thus, the physical prototype also contains two customer areas to map transport chain processes (Fig. 5).[8]

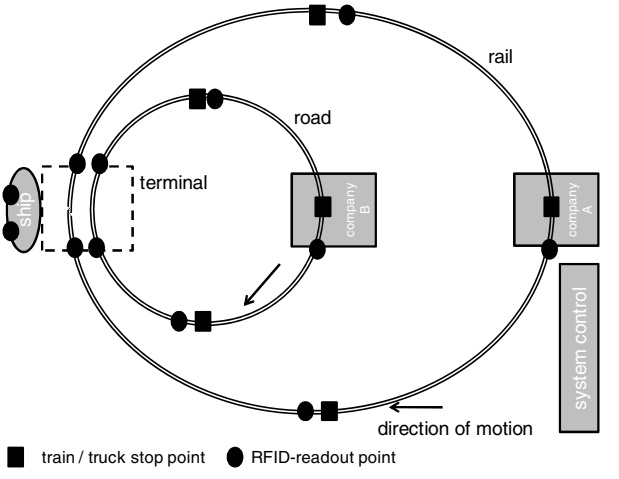

Fig. 5. Physical miniature prototype – schematic overview

The idea is to provide a handy example to the practitioners along the logistic transport chain and to map cross-company processes. This includes tracing and tracking of goods and transport vehicles, exception handling in case of unexpected behaviours of transport vehicles and containers. The goal is also to demonstrate how service oriented computing facilitates cross-company communication (e.g. with automated messages via email/HTTP/SOAP interfaces), increases the amount of valuable information in the transport chain and the ability to easily change processes for different (or the same) customer. SOA further allows developing a high-level business process and defining inputs and outputs for every process step to split a big process in small entities that can be implemented by wrapping service - that wraps legacy system functions - or utility services - that implement small parts of the business logic and are potentially reusable.

References

1. International Organization for Standardization, `http://www.iso.org/iso/home.htm`
2. DIN ISO 668, Series 1 Freight Containers - Classification, Dimensions and Ratings (October 1999)
3. ISO 10374.2, Freight Container – Automatic Identification (2009)
4. ISO 6346, Freight Containers – Coding, Identification and Marking (1995)

[8] Note: This second development phase starts in October 2010 and is planned to be finished within two months.

5. ISO 18186, Freight containers – RFID cargo shipment tag (2010)
6. ISO 17712, Freight Containers – Mechanical Seals (2006)
7. ISO 18185-1, Freight Container – Electronic Seals – Part 1: Communication Protocol (2007)
8. ISO 18185-2, Freight Container – Electronic Seals – Part 2: Application Requirements (2007)
9. ISO 18185-3, Freight Container – Electronic Seals – Part 3: Environmental Characteristics (2007)
10. ISO 18185-4, Freight Container – Electronic Seals – Part 4: Data Protection (2007)
11. ISO 18185-5, Freight Container – Electronic Seals – Part 5: Physical Layer (2007)
12. Hellström, D.: The cost and process of implementing RFID technology to manage and control returnable transport items. International Journal of Logistics Research and Applications 12(1), 1–21 (2009)
13. Logistics-Initiative Hamburg, http://www.hamburg-logistik.com
14. HWF – Hamburg Business Development Corporation, http://www.hamburg-economy.de/index_en.html
15. Becker, T.: Prozesse in Produktion und Supply Chain Optimieren. Springer, Berlin (2005)
16. UN/EDIFACT Message BAPLIE Release: 08A, Bayplan/stowage plan occupied and empty locations message, http://www.unece.org/trade/untdid/d08a/trmd/baplie_c.htm
17. UN/EDIFACT List of all Messages Release, http://www.unece.org/trade/untdid/d00a/trmd/trmdi2.htm
18. Will, T., Blecker, T.: Precarriage in Container Logistics – Analysis and RFID-Driven Modifications in Transshipment Processes. In: 18th IPSERA Conference: Supply Management – Towards an Academic Discipline?, pp. 1554–1569 (2009)
19. Will, T.: Creating a Dynamic Speech Dialogue: How to implement dialogue initiatives and question selection strategies with VoiceXML agents. VDM Verlag Dr. Müller (2007)
20. Erl, T.: Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
21. W3C: Web Service Definition Language (WSDL), http://www.w3.org/TR/wsdl