Adapting a Generic Data Synchronisation Framework for YAWL to Access Clinical Information Systems at the Task Level

Holger Meyer[1], Sebastian Schick[1], Jan-Christian Kuhr[2], Andreas Heuer[1]

[1] University of Rostock {hme,schick,ah}@informatik.uni-rostock.de,
[2] GECKO mbH Rostock jan-christian.kuhr@gecko.de

Abstract. Based on a generic extensible data access framework (DAF) [10], we devised a domain-specific extension to support interoperability of YAWL workflow cases with hospital information systems (HIS). In the scenario considered, the HIS is the principal system that keeps master data and clinical data of the patient. Data transfer between HIS and the YAWL runtime environment is facilitated via exchange of standardized HL7 messages. The solution presented supports read and write task-level synchronization of workflow variables using the patient's case ID as correlation parameter. A first proof-of-concept has been carried out in a real clinical setting.

Keywords: Workflow, Data Access, YAWL, Healthcare, Perioperative Process

1 Introduction

Clinical value chains are in general well-structured and follow an a-priori known execution path. Thus, such scenarios lend themselves to be supported by process aware information systems in general and by business process management systems (BPMS) in particular.

When a BPMS approach is adopted, particular challenges arise if the process engine is external to existing hospital information systems. Patient's master data and visit-related clinical data such as diagnosis and scheduled treatments are usually handled within a HIS, which is the principal system. However, a subset of data is required to be known by the process execution engine to support execution of patient's workflow cases. Conversely, it may be necessary to map process runtime data, such as the duration of activities, back to the HIS. These scenarios call for a data synchronisation solution that integrates HIS with the process execution environment.

In the PERIKLES project we have introduced an extendable generic data synchronisation framework (DAF) [10] and then specifically adopted to the domain of clinical healthcare. The framework helps to avoid inconsistencies within redundantly maintained data and supports transactional aspects within the process and data perspective. Therefore, a layered architecture is used with facilities

to access external data sources, to associate the control-flow perspective with transactional properties like isolation, serializability and recovery.

A second extension to YAWL is the *FlexY* approach [11, 12], which extends the control-flow perspective of YAWL with new concepts for handling process adaptation at run-time. *FlexY* combines the method of late modeling with declarative concepts and underspecification. The runtime adjustment of sub-processes is triggered at certain points of the process according to external data provided by the data access framework. Why flexibility is important and how it can deployed profitably in healthcare is described elsewhere [9, 2].

In this paper we focus on the DAF approach to extend YAWL to handle task-level read and write access to hospital information systems by exchanging standardized messages. The extension described in this paper makes use of YAWL concepts like workflow engine level extensibility (gateway mechanism), automated tasks provided as services, tasks and sub-net variables and parameters based on an XML type system, flexible enactment of sub-processes and exception handling.

2 A Generic Data Synchronisation Framework

The current version of the YAWL engine already provides a simple interface to populate task and net variables using a data source called *data gateway*. In case of using a data source some implementation effort is necessary, because YAWL provides no standard gateways. To provide a configurable, policy based integration of external data sources, we presented a framework called *D*ata *A*ccess *F*ramework in [10]. First, we give a short overview of the main framework components. Afterwards, we present the extensions of the framework which were developed.

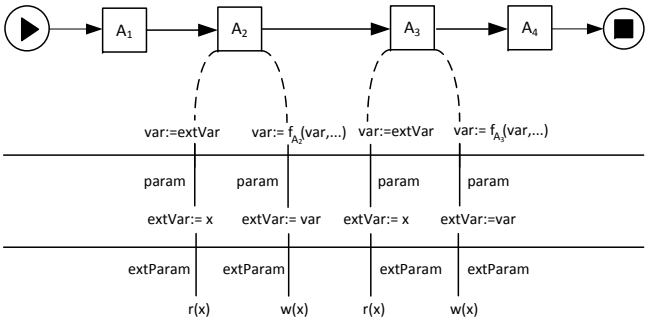

Fig. 1. Data access on external data sources in YAWL

External Variables Access to data in external sources should be transparent to the user who models and executes a YAWL process model. Therefor, we intro-

duce a new variable type *external variable* in YAWL [10]. An external variable defines a view on a data source by specifying the data source and required configuration parameters. Additionally, each data source is encapsulated by a plug-in implementation. The plug-in integration is outlined in the following section.

Figure 1 shows the main idea behind the external variable concept. Each variable (c.f. Fig. 1: *var*) in a process model, can be bound to an external variable (c.f. Fig. 1: $extVar$), using a parameter mapping (c.f. Fig. 1: $param$). To bind an external variable (c.f. Fig. 1: $extVar$) to an plug-in, we use a new type of parameter called external mapping (c.f. Fig. 1: $extParam$). For each net we can define a set of external variables with the type Ext, a set of plug-in id's $PlugInID$ and a set of external parameter mappings $extParamID$.

Definition 1 (External parameter mapping). *A mapping is defined as* $(pID; distKey; map; rp; wp)$*, with:*

- pID *references a plug-in which should be used,*
- $distKey$ *is the key attribute to identify a single XML fragment,*
- map *is an XQuery query which defines the transformation rule between an external variable and the respective data source (with the ID* pID*),*
- rp *is a local read policy (consistent, read-only) for the external variable and*
- wp *is a local write policy (consistent, immediate) for the external variable.*

In Def. 1 an external parameter mapping is defined, where pID is used to identify a plug-in. The $disKey$ and map parameters describe a data item in the source and the rp and wp parameters are used to define the access policies.

Framework Architecture To support external variables within YAWL, we extend the existing YAWL data gateway implementation with our own *data access gateway* [10]. However, in this scenario we use a simplified version which do not use extended transactional services. We use a data gateway which encapsulates the data access framework and manages the access to different plug-in implementations. A Data Source Manager configures further processing using the variable mapping of the external variable and selects the plug-in which have to be invoked.

3 Adaptation to Clinical Information Systems

In healthcare, interoperability between clinical information systems is generally accomplished via exchange of standardized HL7 messages. HL7 messages are made up of segments, fields and delimiters. The *MSH* segment in the example (cf. Fig. 2) indicates that the message is of type ADT (admission, discharge, transfer) and has been caused by an *A01* (patient admission) trigger event.

3.1 HL7 Plug-in

Supporting clinical healthcare workflows by YAWL requires integration of given hospital information systems with the process execution runtime. Following our

```
MSH|^~\&|SAP-ISH|ROD|ITB265||20101123025931|NP11I0|ADT^A01|0000688151|P|2.1
EVN|A01|20101123025000
PID|||265029061|265051944|Nachname001^Vorname001|Mädchenname001|19350128|F|||Musterstr1.
001^^Musterort001^^66976^DE||00000 SELBST||D|W|02
PV1|00001|I|26513.2^^^2651IA|NO|||||1A|||||||||||NO|||||||||||||||R|2651964931|||||20101123025000
```

Fig. 2. Example of an HL7 message

general approach, we have thus extended the existing DAF by an HL7-plug-in to support read/write access of YAWL processes to HL7 enabled clinical systems. The basic architecture of our approach is shown in Fig. 3.

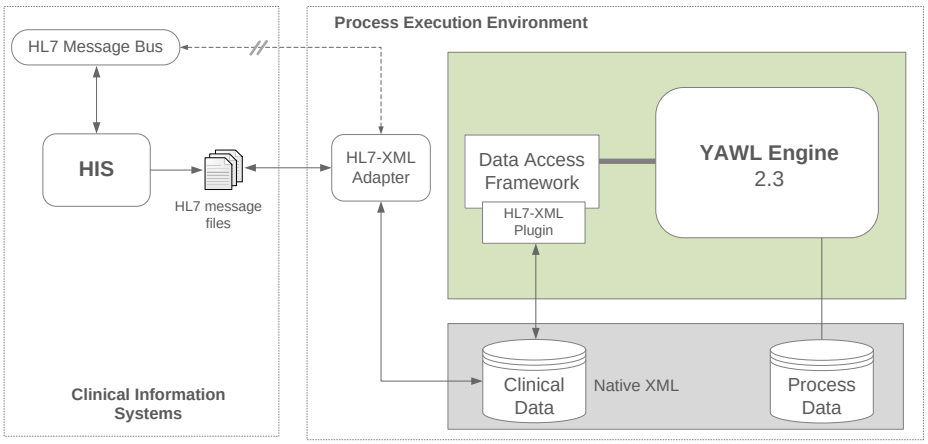

Fig. 3. General architecture of the HIS - YAWL integration

Extending the Process Specification By utilizing extended attributes one can specify at the task level data synchronization policies for individual variables. These policies instruct the DAF about the mode of the data access (read or write), the correlatable parameter to be used (e.g. the patient's medical case id), the plug-in that handles the request, and an XPath expression that extracts the relevant data from the XML representation of the HL7 message. By calling the plug-in the data synchronisation with HL7 sources is handled for all *external* variables.

Listing 1 shows an example of a definition of an external variable. Besides a plug-in ID (`pluginID=XML_PostgreSQL_ADT_Extract`), the source description is defined by an SQL table name (`xmlColumn=patientdata`) together with key attributes (`mappingKey=medcaseid,msgtype,datetime`) and the selection of an XML fragment (`mapping=//patientdata/pv1/patientclass`). We also define some initial values for two of the key attributes (`mappingKeyVal=,A01,[max]`) and a table for write access (`writeProcess=createHl7[public.hl7out;m1]`).

To identify HL7 messages in the database, a number of different attributes are necessary, which are used as key attributes. Some of the key attributes are known at design time, like *message type* or *aggregate functions* over an attribute. Other attributes are only known at runtime, like *medical case ID*. These compound primary keys are described by the *mappingKey* attribute.

Listing 1. External variable definition

```
extVar_pluginID=XML_PostgreSQL_ADT_Extract
extVar_readPolicy=readOnly
extVar_xmlColumn=patientdata
extVar_mapping=//patientdata/pv1/patientclass
extVar_mappingKey=medcaseid,msgtype,datetime
extVar_mappingKeyVal=,A01,[max]
extVar_writeProcess=createHl7[public.hl7out;ml]
```

In most cases HIS assume the role of a data source. Upon the occurrence of predefined events the HIS emits HL7 messages that travel along a message bus or may be exported to the file system. In order to validate our approach in a real clinical setting without running the risk of interfering with production systems, we have used the latter method (cf. Fig. 3).

Since YAWL handles task and case variables as XML data types, every *incoming* HL7 message needs to be transformed into an XML representation that is compliant to the XML schema of the YAWL process specification. Conversely, all *outgoing* messages must be converted from XML to a proper HL7 representation. Therefore, depending on whether variables are specified for read or write access, we use different tables for read and write access. We therefore extend the source definition of an external variable, such that a second source for write access can be defined (cf. Lst. 1: `extVar_writeProcess`). These bi-directional message transformations are accomplished by an HL7-XML adapter (cf. Fig. 3). In read mode, the adapter polls the HL7 export file for new messages and transforms any of these into an XML representation. In order to avoid excessive transformations each time a message is accessed, all XML-formatted messages are stored persistently in a database (cf. Fig. 3: Database HL7Messages).

3.2 Proof of Concept

The validity of our approach for real clinical settings has been demonstrated by a single-day proof of concept (PoC) deployment in a German hospital. Figure 4 shows the principal design. While the PoC has been restricted to *read-only* synchronization, the figure shows also *write-mode* use cases that may be relevant for future evaluations. The underlying clinical scenario is a perioperative careflow, which, for the sake of clarity, has been greatly simplified.

From the perspective of the **treatment layer**, each patient follows a sequence of events, beginning with the admission to the hospital end ending with the post-operative transport to the intensive care unit (ICU). While this sequence is strongly aligned with the process layer, there are also important interactions with the hospital information system.

The **HIS layer** may be viewed as consisting of two distinct components: The patient management system (PMS) keeps the patient's master data such as name, address, gender, and date of birth as well as the visit-related *administrative* case data. For every visit a patient may make to the hospital, the PMS assigns a medical case ID to this visit. On the other hand, the clinical information system (CIS) is responsible for recording and managing the *medical* visit-related data, such as diagnosis, lab test results, and the planned treatment. Both subsystems are capable of emitting and receiving HL7 messages in order to communicate business events to other information systems or to process such events that have been emitted by other systems. HL7 messages may contain administrative data only (e.g. in case of an admission or transfer event) or primarily medical information (e.g. in case of communicating lab results).

The **process layer** has been implemented by the YAWL runtime environment and is thus external to the hospital information systems. The use case evaluated in the PoC required a read-mode synchronization of the workflow instance with the patient's master data as well as the visit-related administrative data, both of which are kept inside the PMS. Upon start of a workflow case, the patient's medical case ID is passed to the process as a parameter. The medical case ID, which is to be distinguished from the workflow engine's case ID, serves as the principal correlation link between the process layer and the clinical information systems. Once the task *Supply Master Data and Case Admin Data* becomes enabled, synchronization with the HIS layer takes place in read mode such that the required administrative data are extracted from an appropriate HL7 message that has been emitted by the PMS. This information may then be reviewed by the clinician that is executing the task before continuing with *Do Surgical Assessment*. In this way, synchronization relieves the stakeholders of re-entering the same data twice.

Possible Future Use Cases Data synchronization between process and HIS layer is by no means restricted to the read-only mode. Our approach is also capable of supporting write-mode scenarios that may be interesting from the clinical point of view and that lend themselves for future evaluation. In the figure, we give two motivating examples. In both of which, the workflow case acts as information source and a hospital information system as recipient of that information. Compared with the former scenario, the situation is thus reversed.

The workflow tasks *Transfer To OR* and *Transfer To ICU* correspond to events that are relevant from an *administrative* point of view. So rather than capturing this information twice, it may be desirable to run a write-mode synchronization on completion of these tasks, respectively, to communicate the time stamp of the event to the PMS. In terms of the HL7 messaging concept, this would also amount to automatically executing a *transfer patient* request inside the patient management system.

The task *Do Surgical Procedure* may have captured *clinical* data such as duration and description of the surgical procedure. However, this information may also be required by the CIS. Hence, one could think of enabling the workflow

system to write this information back to a HIS layer by sending an appropriate HL7 message to be consumed by the clinical information system.

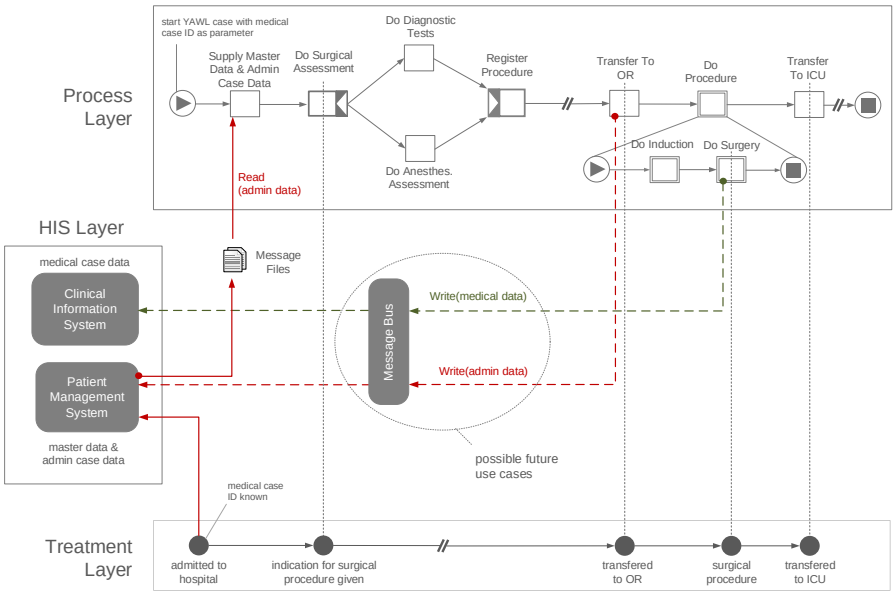

Fig. 4. Simplified clinical scenario of the proof of concept. The figure shows a use case that has been evaluated as in a real clinical setting (solid red line) as well as possible future uses cases (dashed red and green lines).

4 Related Work

YAWL is used and extended by several research initiatives including but not limited to clinical workflows. The framework supports the data patterns defined by Russel et al. [8]. The *Proclet* approach [5] extends YAWL to use process fragments, which can communicating via channels and ports. The support of different types of flexibility was demonstrated in several frameworks. The *Worklet* approach [1] extends YAWL with a concept of late-binding process fragments. In [3] an extension to YAWL implementing configurable process models is presented.

In the context of data-driven processes, different approaches for data integration exist. *SIMPLE* [7] describes strategies for accessing external data sources, by providing an abstraction layer for data management. Frameworks like *PHILharmonicFlows* [4] or *Corepro* [6] provide an extensive integration of object behavior, object interactions and process execution within the process model.

5 Conclusion

In this paper, we present an adaptation to a data synchronisation framework for YAWL and its application to a clinical information system. We showed that an efficient way to access data from different clinical information systems during runtime is necessary. We then adapted an existing data synchronisation framework to support accessing these data at the task level. In future, we intend to use YAWL in different scenarios supporting also flexibility.

Acknowledgments Part of this work has been funded by the German Federal Ministry of Education and Research (BMBF) under grant 01IS099009. We thank Markus Bandt and Viktor Baidinger for their contribution to the implementation. Validation of the key concepts has been done in cooperation with the Hetzelstift hospital in Neustadt/Weinstraße, Germany. The authors wish to thank especially Dierk Vagts and Bernd Wössner for supporting the proof-of-concept in a real clinical setting.

References

1. M. Adams, A. ter Hofstede, D. Edmond, and W. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *OTM Conferences*, pages 291–308, 2006.
2. M. Bandt, R. Kühn, S. Schick, and H. Meyer. Beyond flexibility — workflows in the perioperative sector of the healthcare domain. *Electronic Communications of the EASST*, 37:146–157, 2011.
3. F. Gottschalk, W. van der Aalst, M. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221, 2008.
4. V. Künzle and M. Reichert. Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.
5. R. S. Mans, N. Russell, W. van der Aalst, P. Bakker, A. Moleman, and M. Jaspers. Proclets in healthcare. *Journal of Biomedical Informatics*, 43(4):632–649, 2010.
6. D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *CAISE*, pages 48–63, 2008.
7. P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, and F. Leymann. Simpl - a framework for accessing external data in simulation workflows. In *BTW*, pages 534–553, 2011.
8. N. Russell, A. ter Hofstede, D. Edmond, and W. van der Aalst. Workflow data patterns: Identification, representation and tool support. In *ER 2005*, volume 3716 of *LNCS*, pages 353–368. Springer Berlin / Heidelberg, 2005.
9. S. Schick, H. Meyer, M. Bandt, and A. Heuer. Enabling yawl to handle dynamic operating room management. In *BPM Workshops (2)*, pages 249–260, 2011.
10. S. Schick, H. Meyer, and A. Heuer. Enhancing workflow data interaction patterns by a transaction model. In *ADBIS*, pages 33–44, 2011.
11. S. Schick, H. Meyer, and A. Heuer. Flexible publication workflows using dynamic dispatch. In *ICADL*, pages 257–266, 2011.
12. S. Schick, H. Meyer, and A. Heuer. FlexY: Flexible, datengetriebene Prozessmodelle mit YAWL. In *BTW*, pages 503–506, 2013.