Monitoring for Digital Preservation of Processes

Martin Alexander Neumann[a], Till Riedel[a], Philip Taylor[b],
Hedda R. Schmidtke[a], and Michael Beigl[a]

[a]Karlsruhe Institute of Technology, Karlsruhe, Germany
{mneumann,riedel,schmidtke,beigl}@teco.edu
[b]SAP Research, Belfast, United Kingdom
{philip.taylor}@sap.com

Abstract. Digital Preservation is an important challenge for the information society. Reliable information and communication technology is crucial for most companies and software failure, is a considerable risk. Use of technologies such as Software as a Service (SaaS) and Internet of Services (IoS) means that business processes are increasingly supported by distributed, service oriented systems. We propose a concept and methods for capturing of contextual information, event causality and timing for Digital Preservation of distributed business processes and services. The architecture is derived from an architecture for monitoring sensing systems. We add a reasoner that can check whether processes adhere to explicit contracts and detect behavior anomalies, and we sketch how an inductive learner can be used to detect anomalies not covered by these contracts.

Keywords: Digital preservation, context monitoring, sensing systems

1 Introduction

The main problem of digital storage, in contrast to traditional media such as paper or stone, is that specific computing technologies are needed to access it. Reading devices and formats change at a rapid speed that makes it difficult to restore information. The goal of Digital Preservation (DP) is the preservation of information that is stored digitally. DP of business processes and services is a novel and important challenge for the information society. Reliable information and communication technology is crucial for most companies, and software failure is a considerable risk. The presence of Software as a Service (SaaS) and Internet of Services (IoS) means business processes are increasingly supported by service oriented systems where numerous services provided by different providers located in different geographical locations are composed to form service systems which will continue evolving. Besides the advantages of SaaS and IoS, there is the danger of services and service providers disappearing (for various reasons) leaving partially complete business processes. DP of business processes and services therefore requires that the set of activities, processes and tools that ensure continued access to services and software necessary to produce the context within which information can be accessed, properly rendered, validated and transformed into knowledge can be preserved. Means for DP are proposed for several layers of the network. *Software contracts and service contracts* protect the links between pieces of code with a formally verifiable interface [1]. *Conventional DP and digital archiving* ensure that data can be restored, even when the original storage technology is out of use [5]. *Virtualization* allows a nearly complete conservation of computing environments [4]. *Digital escrow services* ensure that software systems can survive their providers [7].

However, these methods do not yet take context into account. Often, it is not clear at a given time, which information would be required later to restore a system. What is a clear boundary of technical feasibility today can be a parameter to be represented in the future. Computing power is a straightforward example. Consider a time t_0 where we have a network connection whose speed is below the speed of a database lookup. Assume there is a race condition: our service internally sends a request to a remote server and concurrently looks up a required parameter in the database. As the database access is always faster than the network request, we obtain the parameter before the remote server response. If we later (time t_1) restore the digitally preserved system in a virtual environment, we might obtain a system in which the network is faster than the database leading to different results than with the original system. Capturing such constraints is therefore crucial. However, we often do not understand the parameters that make up these constraints. The context in which things happen is usually implicit, informal and natural to the actors at time t_0. It is something that they take as a constant, given information to which one does not need to pay attention. It becomes a relevant parameter that has to be modeled only for the actors at time t_1, to whom this information is no longer constant and given.

An important challenge is therefore to capture contextual information for preservation. Parameters of context that have to be preserved include timing constraints and causal links, i.e. parameters that are often at least partially constrained through Service Level Agreements (SLA) or service contracts, but also, e.g., the location of servers, which determine e.g. timezones and applicable law, as well as the involved natural or legal persons. Such parameters of context can be preserved if changes in these parameters can be detected: conventional debugging techniques can be used to monitor such changes for the formalized parts of the system. However, these tools cannot yet handle the full complexity of a system operating in an at least partially unpredictable environment. Advances on this challenge have been made in the area of debugging of sensing systems.

Debugging approaches based on analysis of a system's run time enable to monitor processes, as they have the ability to inspect the entire state of a system. Debugging approaches may either operate on a physical system or on a simulated one. Simulations differ in the degree of simulation accuracy and they offer high visibility into, and control on the run time state at high performance in contrast to physical systems. In addition, physical environments serve as the foundation for debugging techniques which for example offer entire system state inspection and step debugging [14]. They can for example be based on controlled testbed facilities which enhance the visibility into, and the control on networks and nodes. Tracing-based debugging techniques primarily refer to discrete event tracing, whereby events may, for example, be network messages [3] or events generated on sensor nodes [11]. Analysis on gathered traces may serve for fault detection only, or additionally offer isolation and identification of faults [12]. Program flow tracing is another type of tracing approach to address debugging [2]. Model-based approaches that continuously monitor systems to supervise formally-given knowledge on system behavior are closest to our approach. Models may for example be based on automata [6], Petri Nets (PNs) [13] or logic [8]. Our approach uses PN-based models which are given either explicitly or implicitly learned from informal knowledge.

We propose a concept and methods for capturing and monitoring such contextual information. Our architecture is based on an architecture for monitoring sensing systems (Sect. 2.1). We complement this with a deductive reasoner (Sect. 2.2) that can check whether processes adhere to explicit contracts to detect behavior anomalies, and we sketch how an inductive learner can be used to detect anomalies not covered by these contracts.

2 Monitoring of Processes in Context

Monitoring of process contexts provides the abilities to decide *when* to preserve *what* elements of a process (or the entire one) digitally, based on the supervision whether context behavior adheres to given contracts. Contracts on contexts can either be given *explicitly*, e.g. based on SLAs, or *implicitly*, based on learned normality behavior. Primarily, collected contexts are subject to be contracted on, i.e. the values that define contexts. Secondarily, the context behavior over time are contracted on based on their *causality* and *timing*.

We differentiate two types of processes: *legacy processes* which are not enabled for DP yet, and *DP-enabled processes* which have been explicitly designed for preservation. The difference between both types is that DP-enabled processes require defined process semantics, e.g. based on BPMN, and defined relevant process contexts – both of which may not available with legacy processes.

Legacy processes can be extended to learn the normality behavior of their relevant contexts and to supervise whether the contexts adhere to learned normality in future. This extension fosters anomaly detection in established processes and restored ones. Detected anomalies either reflect faults or deliberate process changes that indicate a new version of a process. Therefore, anomalies represent events to determine whether a new version of a process should be preserved, e.g. by the help of a DP engineer. The extension of legacy processes limits qualitative evaluation of anomalies (whether they reflect failure or deliberate change), because the extension lacks fine-grained association of anomalies to process elements with respect to process semantics. This association would provide qualitative evaluation of anomalies and DP at process element level.

This fundamental anomaly detection will also be available with future DP-enabled processes. In addition, DP-enabled processes will supersede this anomaly detection by offering fine-grained association of anomalies to process elements and qualitative evaluation of anomalies and DP at process elements scale. Contexts of DP-enabled processes will be well understood and defined at design time of processes. This enables explicit contracts consisting of rules on process contexts whose behaviors are well-defined with respect to process semantics, e.g. based on rules derived from the semantics.

2.1 Distributed State Inspection

The DSI concept offers to inspect sensor nodes in an entire (physical, simulated, mixed-mode) WSN and collect events sourced by the nodes, e.g. notifications of state changes. The concept is based on a 3-layer architecture shown in figure 1, of which the lower two provide state inspection and event tracing.

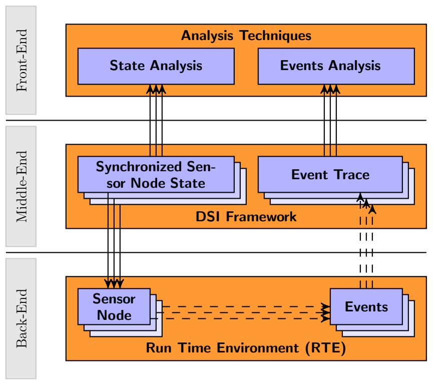

Fig. 1. DSI Architecture Overview

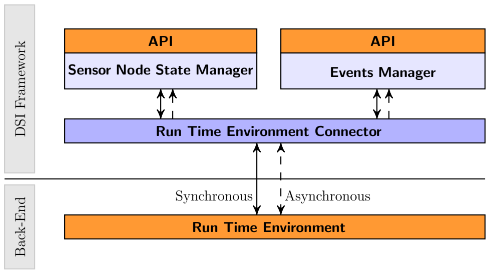

Fig. 2. Focus on DSI Framework

The *back-end* is formed by the Run Time Environment (RTE) for any programs on a WSN. The RTE provides the state of all running programs to clients. In addition, the RTE emits discrete events, e.g. in case a state on a sensor node changed. Events are emitted via an asynchronous interface to which clients can subscribe. The state inspection interface allows clients to connect to the RTE and perform state synchronization operations whose execution is performed synchronously between clients and the server. The *middle-end* is represented by a framework that encompasses the modules that enable sensor node state inspection and event tracing to *front-ends*. The framework asynchronously gathers events sourced by the RTE and continuously synchronizes the state of all sensor nodes in an asynchronous fashion after registering for relevant state changes on sensor nodes in the RTE.

The *DSI framework* is meant to be a foundation for analysis approaches (in the *front-end*) that analyze the distributed state of and discrete events from a WSN. From a client's perspective, the DSI framework offers a set of APIs to continuously synchronize a model of the RTE as illustrated in figure 2. The framework is founded on the *RTE Connector* which acts as a proxy to the shielded RTE, establishing a loose-coupling (integrated but isolated) between the RTE and the other components of the framework.

Using events from the RTE, the *state manager* is notified of relevant state changes. Furthermore, the *events manager* organizes events from the RTE in general. Clients subscribe to event types in the events manager whereby the manager provisions the event gathering and notification of registered clients.

The DSI framework requires the RTE to offer the following features: (1) inspect the state of the programs in a WSN; (2) event generation; and (3) a notification mechanism for relevant state changes (clients register for notifications of relevant state changes via discrete events).

2.2 Context Engine

We extend the DSI concept by a *context engine* (depicted in figure 3), which is built on top of the DSI framework and provides the illustrated monitoring and anomaly detection capabilities. DP of software services requires dedicated reasoning systems, which, given a set of facts monitored with the monitoring system can determine: (1) *SLA compliance queries* asking whether the monitored set of facts is consisted with service level agreements (SLA); (2) *SLA preservation queries* asking whether the monitored set of facts fulfills the requirements of a DP case specified between the parties in an SLA; (3) *Anomaly queries* asking whether the monitored set of facts is consistent with previous behavior for detecting whether there might have been an unnotified change of operations. The idea is that the DSI framework enables monitoring contexts by mapping the values of contexts into the program state on the sensor nodes, effectively synchronizing a network-wide context representation of the RTE into the DSI framework. The context monitor translates the network-wide contextual information into a logical format to provide a logical representation of every monitored context change.

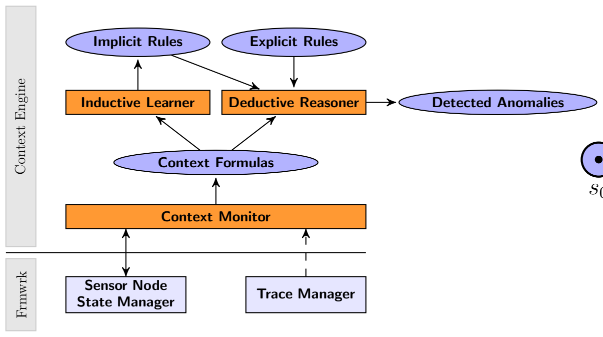

Fig. 3. Context Engine Architecture

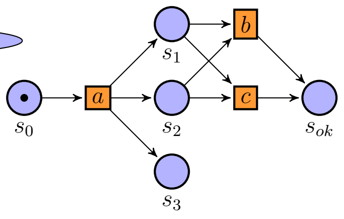

Fig. 4. CL Example as Petri Net

The architecture contains an *inductive learner* and a *deductive reasoner*: the queries (1) and (2) can be decided using a classical deductive reasoning system. The third type of query additionally needs a behavior learning mechanism. The deductive reasoner detects anomalies on context behavior. Its operations are based on deduction of contradictions from the context formulas by the context monitor, the implicit rules and explicit rules. A deducted contradiction results in an anomaly detection event, as the monitored context behavior contradicts the one which is formally stipulated by the rules (implicit and explicit ones). We suggest a mechanism based on Context Logic (CL), a formalism that features a syntax similar to Description Logics (DL) with a specific set of relations employed to describe contextual relations [10]. A detailed exposition of CL is beyond the scope of this paper and can be found in [9].

Syntax	Example	Reading
$c \sqsubseteq_{\text{who}} d$	paul $\sqsubseteq_{\text{who}}$ admin	c is a sub-group of d
$c \sqsubseteq_{\text{what}} d$	car $\sqsubseteq_{\text{what}}$ smart-vehicle	c is a sub-class of d
$c \sqsubseteq_{\text{when}} d$	July.1.13:00 $\sqsubseteq_{\text{when}}$ Summer	c is during d
$c \sqsubseteq_{\text{where}} d$	geo4154N 1230E $\sqsubseteq_{\text{where}}$ Rome	c is (spatially) in d

Table 1. Relations in Context Logic

Intuitively, atomic formulae of CL describe partial ordering relations between contexts, which can be atomic identifiers or complex expressions. In contrast to DL, the atomic formulae of CL can be combined with propositional logic conjunctives. Four fundamental relations of CL are shown in table 1. However, using additional relations $\sqsubseteq_{\text{how}}$, $\sqsubseteq_{\text{why}}$, basic condition/event structures as required for the DSI can be encoded. A condition/event Petri Net (PN) $\langle P, T, F, m_0 \rangle$ consists of a set of *places* P encoding *conditions* and a set T of *transitions* encoding *events*, where $F \subset (P \times T) \cup (T \times P)$ is the set of edges of the net and m_0 is the *initial marking*. Here a function $m_i : P \to \{0, 1\}$ is called a *marking*. A transition t is activated in a marking m_i iff for all p holds:

- if $(t, s) \in F$ then $m_i(p) = 0$
- if $(s, t) \in F$ then $m_i(p) = 1$.

We match the how-dimension to conditions, i.e. places, and the why-dimension to events, i.e. transitions. We can encode the edges in the net shown above (Fig. 4) with $c(i)$ representing the current context at time i by encoding pre-conditions and post-conditions:

$$[s_0 \sqsubseteq_{\text{how}} c(i)] \wedge \neg[s_1 \sqcup s_2 \sqcup s_3 \sqsubseteq_{\text{how}} c(i)] \to [c(i) \sqsubseteq_{\text{why}} a] \quad (1)$$

$$[s_1 \sqcup s_2 \sqsubseteq_{\text{how}} c(i)] \to [c(i) \sqsubseteq_{\text{why}} b] \oplus [c(i) \sqsubseteq_{\text{why}} c] \quad (2)$$

$$[c(i) =_{\text{why}} a] \to [s_1 \sqcup s_2 \sqcup s_3 \sqsubseteq_{\text{how}} c(i+1)] \quad (3)$$

$$[c(i) =_{\text{why}} b] \oplus [c(i) =_{\text{why}} c] \to [s_{ok} \sqsubseteq_{\text{how}} c(i+1)] \quad (4)$$

- If s_0 holds in $c(i)$ and s_1, s_2, and s_3 do not hold, then a fires in $c(i)$.
- If s_1 and s_2 hold in $c(i)$ and s_1, s_2, and s_3 do not, then c xor d fires in $c(i)$.
- If a fires in $c(i)$, then s_1, s_2, and s_3 hold in $c(i + 1)$.
- If b xor c fires in $c(i)$, then s_{ok} holds in $c(i + 1)$.

3 Conclusion

We proposed a concept and methods for capturing contextual information, event causality and timing for DP of distributed business processes and services, which can track changes and generate decision events, so as to automatically trigger basic preservation steps or to notify a DP engineer. The architecture is derived from a concept for monitoring sensing systems. We added a deductive reasoner that can check processes for adherence to explicit contracts, and we sketch how an inductive learner can be used to detect anomalies not covered by these contracts.

Acknowledgments The authors would like to acknowledge the funding by the European Commission under the ICT project "TIMBUS" (Project No. 269940, FP7-ICT-2009-6) within the 7th Framework Programme.

References

1. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In: Proceedings of CONCUR 2010. pp. 162–176. Springer (2010)
2. Cao, Q., Abdelzaher, T., Stankovic, J., Whitehouse, K., Luo, L.: Declarative tracepoints: a programmable and application independent debugging system for wireless sensor networks. In: Proceedings of SenSys 2008. pp. 85–98. ACM (2008)
3. Chen, B.R., Peterson, G., Mainland, G., Welsh, M.: Livenet: Using passive monitoring to reconstruct sensor network dynamics. In: Proceedings of DCOSS 2008. pp. 79–98. Springer (2008)
4. Krsul, I., Ganguly, A., Zhang, J., Fortes, J.A.B., Figueiredo, R.J.: VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In: Proceedings of SC 2004. IEEE, Washington, DC, USA (2004)
5. Lee, K.H., Slattery, O., Lu, R., Tang, X., Mccrary, V.: The state of the art and practice in digital preservation. Research of the National Institute of Standards and Technology 107(1), 93–106 (2002)
6. Li, P., Regehr, J.: T-check: bug finding for sensor networks. In: Proceedings of IPSN 2010. pp. 174–185. ACM, New York, NY, USA (2010)
7. Nycum, S.H., Kenfield, D.L., Keenan, M.A.: Debugging software escrow: Will it work when you need it? Computer Law 4(3), 441–463 (1984)
8. Römer, K., Ringwald, M.: Increasing the visibility of sensor networks with passive distributed assertions. In: Proceedings of REALWSN 2008. pp. 36–40. ACM, New York, NY, USA (2008)
9. Schmidtke, H.R., Hong, D., Woo, W.: Reasoning about models of context: A context-oriented logical language for knowledge-based context-aware applications. Revue d'Intelligence Artificielle 22(5), 589–608 (2008)
10. Schmidtke, H.R., Woo, W.: Towards ontology-based formal verification methods for context aware systems. In: Tokuda, H., Beigl, M., Brush, A., Friday, A., Tobe, Y. (eds.) Pervasive 2009. pp. 309–326. Springer (2009)
11. Tolle, G., Culler, D.: Design of an application-cooperative management system for wireless sensor networks. In: Proceedings of EWSN 2005. pp. 121–132 (2005)
12. Woehrle, M., Plessl, C., Lim, R., Beutel, J., Thiele, L.: Evant: Analysis and checking of event traces for wireless sensor networks. In: Proceedings of SUTC 2008. pp. 201–208. IEEE, Washington, DC, USA (2008)
13. Wu, Y., Kapitanova, K., Li, J., Stankovic, J.A., Son, S.H., Whitehouse, K.: Run time assurance of application-level requirements in wireless sensor networks. In: Proceedings of IPSN 2010. pp. 197–208. ACM, New York, NY, USA (2010)
14. Yang, J., Soffa, M.L., Selavo, L., Whitehouse, K.: Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In: Proceedings of SenSys 2007. pp. 189–203. ACM, New York, NY, USA (2007)