Keeping Pace with the History of Evolving Runtime Models

Lucas Sakizloglou[✉], Matthias Barkowsky[✉], and Holger Giese[✉]

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
<name>.<surname>@hpi.de

Abstract. Structural runtime models provide a snapshot of the constituents of a system and their state. Capturing the history of runtime models, i.e., previous snapshots, has been shown to be useful for a number of aims. Handling, however, history at runtime poses important challenges to tool support. We present the INTEMPO tool which is based on the ECLIPSE Modeling Framework and encodes runtime models as graphs. Key features of INTEMPO, such as, the integration of temporal requirements into graph queries, the in-memory storage of the model, and a systematic method to contain the model's memory consumption, intend to address issues which seemingly place limitations on the available tool support. INTEMPO offers two operation modes which support both runtime and postmortem application scenarios.

Keywords: runtime models · time-awareness · temporal graph queries

1 Introduction to InTempo

A (structural) *Runtime Model* (RTM) provides a snapshot of the constituents of a system and their state [3]. RTMs are typically employed in the context of *Self-adaptive Systems* (SAS) [4], where a feedback loop adapts the system behavior at runtime in response to external or internal stimuli, the latter represented as model fragments in the RTM and detected via the execution of model queries.

Encoding an RTM as a graph enables detection via *graph queries*, which specify a sought (graph) pattern. Such an encoding conforms to a *metamodel* which restricts the structure of model instances and defines types of vertices, edges, and attributes. Formally, these concepts rely on *typed, attributed graph transformation* [6] where graphs are typed over a *type graph*.

Capturing the *history* of RTMs, i.e., previous snapshots, may be useful for a number of aims such as the detection of recurrent behavior or postmortem analysis [3,8]. However, handling history at runtime poses important challenges to tool support. Tools are required to enable the specification and timely execution of queries with *temporal requirements*, i.e., requirements on the evolution of patterns over multiple snapshots. Timely execution is crucial for SAS, where a loop may depend on query results before planning and performing adaptations.

Faced with these challenges, the available tool support is seemingly limited either by the lack of support for direct specification of temporal requirements in graph queries [5] or by the on-disk representation of the model [8,11] that introduces an overhead on execution times in runtime settings, e.g., in SAS.

We present the INTEMPO (Incremental queries with TEMPORal requirements) tool (available online at [13]) which is based on the eponymous querying scheme in [15] and aims at mitigating these limitations. INTEMPO introduces ITQL, a language for the specification of *temporal graph queries*, which allow for the expression of temporal requirements. The core functionality of INTEMPO executes a query over an in-memory RTM which captures information about previous snapshots, called *Runtime Model with History* (RTM^H), and returns the pattern occurrences in the RTM^H that satisfy the specified temporal requirements. INTEMPO is implemented in the ECLIPSE Modeling Framework (EMF) [7] and can be used either via the ECLIPSE user interface or via an API. The latter enables INTEMPO results to be utilized by other tools, e.g., a SAS feedback loop.

INTEMPO offers two operation modes intended for different application scenarios (see Figure 1 for an illustration). The *RTM^H Analysis* (Section 2) constitutes the core functionality of INTEMPO and executes a user-specified ITQL query (in a file with *.itql* extension—required extensions are in parentheses in Figure 1) over a user-provided RTM^H, i.e., a persisted instance of an EMF model (in the standard *xmi* format). This mode returns the query results for the given RTM^H. Query results are kept in-between analyses and are updated by each RTM^H Analysis, which is also known as *incremental (query) execution*. The RTM^H Analysis is intended to be used in settings where query results can be further utilized at runtime. For instance, a SAS feedback loop may use INTEMPO to detect problems formulated as patterns, similarly to [9]. Subsequently, the query results may be utilized to plan adaptations which address these problems.

The *LogAnalysis* operation mode (Section 3) assumes that, instead of being captured by an RTM^H, past and present data about the system have been captured in an event log. INTEMPO introduces E2P, a specification language that allows for the mapping of event types to corresponding modifications of model fragments, i.e., nodes, edges, and attributes. As input, LogAnalysis requires the ITQL query, the log (with comma-separated values), and the E2P mapping. It then processes the log and maintains an internal RTM^H which it uses to perform RTM^H Analysis upon every event. LogAnalysis is intended for postmortem scenarios. Thus, it returns the results that were valid after each RTM^H Analysis sorted by the log timestamps, which affords a global, yet detailed, view on the evolution of the system state.

INTEMPO is capable of containing the data accumulation in the RTM^H by systematically discovering and discarding data that is obsolete with respect to a given timestamp, i.e., not relevant to future query executions—this capability is presented in detail in [15]. Note that an implicit requirement of both operation modes is that the metamodel of the analyzed system has been encoded as an EMF Ecore model and is available in ECLIPSE (gray input in Figure 1).

Fig. 1: INTEMPO Execution Modes and Exemplary Application Scenarios

Exemplary Application To demonstrate the features and operation of INTEMPO we rely on an example drawn from the case-study conducted in [15]. Based on real-world smart medical environments, the case-study envisions a Smart Healthcare System (SHS) where certain medical procedures are automated and performed by devices, such as a smart pump administering medicine or a sensor tracking patient data and diagnoses—as otherwise a clinician would be doing. Data collected from the SHS are aggregated and recorded in medical (event) logs.

Fig. 2: SHS Metamodel

INTEMPO requires a metamodel which has been instrumented such that all nodes have at least two attributes named **cts** and **dts**, which capture the time point of creation, respectively deletion, of the node in the system. As an example, see the metamodel of our SHS in Figure 2. Note that to encode **cts** and **dts** for edges in EMF, the respective edges would have to be modeled as nodes. Technically, an RTM^H is an instance of such a metamodel. See G_3 in Figure 3 for an example based on the SHS metamodel: The RTM^H reflects that a node of type **Sensor** that is attached to the patient with **id=1** has been activated and thus has been added to the SHS at timestamp 3. The sensor **status** reflects that the patient has been diagnosed with sepsis. The value ∞ reflects that a **dts** for this node has not been set, i.e., the node is still present in the modeled system.

2 RTM^H Analysis

This section presents an exemplary query in ITQL which it then uses to demonstrate the RTM^H Analysis. It concludes with technical details.

InTempo Query Language (ITQL) Formally, a temporal graph query q is characterized by a (graph) pattern p and an *application condition* ac , denoted $q = (p, ac)$. A match m corresponds to an occurrence of p in the RTM^H . In order for m to be valid, it must satisfy the ac . ITQL supports the formulation of ac in the *Metric Temporal Graph Logic* (MTGL) [10] which supports operators such as negation ($\neg$), existential quantification ($\exists$), conjunction ($\wedge$), and the *metric*, i.e., interval-based, temporal operators *until* (U_I , where I is a time interval over $\mathbb{R}_0^+$) and *since* (S_I), as well as abbreviations such as *eventually*, i.e., $\Diamond_I \exists n = \text{true} \cup_I \exists n$, where n is a graph pattern and *true* is always satisfied. MTGL also supports the *nesting* of patterns to *bind* graph elements in outer conditions and relate them to inner (nested) conditions, i.e., elements common to two patterns n_1 and n_2 refer to the same element in the RTM^H .

MTGL is able to express real-time properties such as “every patient diagnosed with sepsis, must eventually within 5 time units be given the proper drug” (adjusted from the medical guideline in [14]). In an RTM^H of the SHS, INTEMPO can find violations of the property above by executing the ITQL query $q_1 = (n_1, \kappa)$, with κ the MTGL formula $\neg(\Diamond_{[0,5]} \exists n_2)$ and n_1, n_2 patterns representing a sepsis diagnosis and drug administration respectively. The query searches for matches of n_1 in the RTM^H that satisfy κ , i.e., for patients that,

Fig. 3: Exemplary Medical Log and Corresponding RTM^H G_3 and G_9

although diagnosed with sepsis, did not receive a drug within the designated time. In INTEMPO, each match is associated with a *temporal validity*, i.e., a set of time intervals for which, based on the overlap among the *cts* and *dts* of the matched elements and the interval for which *ac* is satisfied, the match is valid. ITQL also allows for the definition of OCL constraints [12] on sought patterns.

Output The ITQL specification for the query q_1 is shown in Figure 4. Performing RTM^H Analysis for the query q_1 on the RTM^H G_9 of Figure 3 returns one match, since there is indeed no Pump attached to the SHS, i.e., a match for n_2 , within five time units after a Sensor was activated, i.e., a match for n_1 was found. The temporal validity interval $[3, 4]$ is returned together with the match. The match, i.e., violation, is indeed valid only for that interval since after timestamp 4, a match for n_2 starts to exist within five time units of a match for n_1 . If the API of INTEMPO is used, the query returns the match of the n_1 pattern, i.e., the EMF objects, together with the temporal validity. In case INTEMPO is used via the UI it displays a message box in ECLIPSE with the following message: SHS@0[] Sensor@3[status=sepsis][[3,4]]. Note that “@” precedes the *cts* of an object and values within square brackets are attributes of the object.

```
( $\$n1$  , !(true U[0,5] E  $\$n2$ ))

declarations{
  n1{shs:SHS
    s:Sensor
    shs -ownedSensors-> s
    [OCL:"s.status='sepsis'"]}

  n2{shs:SHS
    s:Sensor
    u:Pump
    shs -ownedPumps-> u
    shs -ownedSensors-> s
    [OCL:"u.status='drug'"]}
}
```

Fig. 4: Example query in ITQL

Technical Details For the execution of temporal graph queries, INTEMPO employs the operationalization framework presented in [15]. The framework supports the decomposition of a query into a suitable ordering of simpler sub-queries which is executed bottom-up. The outermost query computes the overall result. For pattern-matching, INTEMPO employs the *Story Diagram Interpreter* from [1] which uses heuristics shown to reduce the pattern-matching effort. INTEMPO provides an XTEXT [2] editor for ITQL which supports completion suggestions for element types and validation of the query syntax.

3 LogAnalysis

This section demonstrates the LogAnalysis operation mode which assumes that data from past states have been captured as events in a log. INTEMPO offers the capability to process the system changes and, upon each change, obtain an updated RTM^H which is then used internally to perform RTM^H Analysis.

Events-to-Patterns (E2P) Specification Language

The mapping of log events (which encapsulate system changes) to prescribed modifications on an RTM^H is facilitated by E2P. An E2P specification consists of mappings between events and actions that should be performed on an RTM^H . E2P supports five actions (formulated as verbs): *adds*, to add a node and optionally assign values to the added node's attributes; *adds-ref*, to add an edge between two nodes; *modifies*, to modify the attribute values of a node; *deletes* and *deletes-ref*, to delete a node, respectively an edge, from the RTM^H . To accommodate linked data, E2P allows for the *indexing* of added nodes so that later events can refer to modifications that have been processed earlier. An example of an E2P mapping from an exemplary log in Figure 3 (left) to the corresponding elements of the SHS is shown in Figure 5. Note that edge types, e.g. *OwnedPumps*, are not depicted in Figure 3.

As an example, the event **drug administration** from the medical log in Figure 3 corresponds to the following changes to the (internal) RTM^H : a **Pump** is created; its attribute **status** is set to “**drug**” and its attribute **id** takes the value of the second field after the event name (expressed by the special **p* token), i.e., the **id** field in the log of Figure 3. By default, the **cts** is set to the value of the event field that is next to the event name, i.e., the **ts** field in Figure 3. The *init* statement is used to initialize the RTM^H and the **cts** of nodes within is set to zero. To increase the readability of specifications, an explicit assignment for the **cts** may be omitted: Unless there is an attribute assignment, the **cts** of all nodes is set to the maximum value supported.

Output LogAnalysis provides a view on the matches *per event timestamp*. Performing LogAnalysis on the query q_1 and the log of Figure 3 would return:

```
@3 SHS@0[] Sensor@3[status=sepsis] [[3,∞]]
@9 SHS@0[] Sensor@3[status=sepsis] [[3,4]]
```

First, the **sepsis diagnosis** event is processed which makes the internal RTM^H be identical to G_3 in the same figure. The query is executed using RTM^H Analysis and returns a match, i.e., violation, since at that moment a match for n_2 does not exist in the graph. The temporal validity is equal to $[3, \infty]$, i.e., the match is valid from time point 3 onward. Next, the **drug administration** event is processed which leads to G_9 . The result of RTM^H Analysis for G_9 is the same as the result described in Section 2.

Technical Details In LogAnalysis the query execution framework monitors the RTM^H for changes and, upon every change, recomputes the matches. Previous matches are kept in-between executions and therefore the query is executed incrementally. Similarly to ITQL, E2P is supported by an XTEXT editor that offers syntax validation and completion suggestions for element types.

```
init:{adds shs:SHS}

"sepsis diagnosis":{
  adds s:Sensor [status="sepsis" id=
    *p2]
  adds-ref shs -ownedSensors-> s}

"drug administration":{
  adds u:Pump [status="drug" id=*p2]
  adds-ref shs -ownedPumps-> u}
```

Fig. 5: E2P Example for SHS

4 Conclusion and Future Work

We presented INTEMPO, an EMF tool which enables the specification and incremental execution of temporal graph queries over a runtime model with history. The latter can be either provided as input or obtained by an event log. INTEMPO stands out from relevant tools owing to the integration of temporal requirements into graph queries, the in-memory representation of the model, and the systematic measures to contain memory consumption despite the accumulation of temporal data. Moreover, INTEMPO offers input editors with features that aim at helping the user, e.g. syntax validation. In the future, besides streamlining INTEMPO, we plan to perform extensive evaluation and comparisons with other tools. Moreover, we plan to explore the utilization of INTEMPO in self-adaptation scenarios where the history of the system is required.

References

1. Barkowsky, M., Giese, H.: Hybrid search plan generation for generalized graph pattern matching. *JLAMP* **114**, 100563 (2020)
2. Bettini, L.: Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd (2016)
3. Bencomo N., Goetz S., and Song H.: Models@ run. time: a guided tour of the state of the art and research challenges. *SoSyM* **18.5** (2019)
4. Brun, Y., Di Marzo Serugendo, G., Gacek, C., Giese, et al.: Software engineering for self-adaptive systems, pp. 48–70. Heidelberg (2009) Springer
5. Búr, M., Szilágyi, G., Vörös, A., Varró, D.: Distributed graph queries over models@run.time for runtime monitoring of cyber-physical systems. *STTT* **22**(1)
6. Ehrig, H., Prange, U., Taentzer, G.: Fundamental Theory for Typed Attributed Graph Transformation. ICGT Berlin, Heidelberg (2004) Springer
7. Eclipse Foundation: Eclipse modeling framework (EMF) (Aug 2020), <https://www.eclipse.org/modeling/emf/>, accessed: 2020-10-11
8. García-Domínguez, A., Bencomo, N., Parra-Ullauri, J.M., García-Paucar, L.H.: Querying and Annotating Model Histories with Time-Aware Patterns. *MODELS*. pp. 194–204 (2019) ACM/IEEE
9. Ghahremani, S., Giese, H., Vogel, T.: Efficient utility-driven self-healing employing adaptation rules for large dynamic architectures. *ICAC* (2017)
10. Giese, H., Maximova, M., Sakizloglou, L., Schneider, S.: Metric Temporal Graph Logic over Typed Attributed Graphs. *FASE*, (2019) Springer
11. Gómez, A., Cabot, J., Wimmer, M.: TemporalEMF: A temporal metamodeling framework. *ER*, vol. 11157, pp. 365–381. (2018) Springer
12. Kleppe, A., Warmer, J.: An introduction to the object constraint language (OCL). In: *TOOLS* p. 456 (2000)
13. MDELab: InTempo Homepage, <http://www.hpi.uni-potsdam.de/giese/public/mdelab/mdelab-projects/intempo/>, accessed: 2021-01-19
14. Rhodes, A., Evans, L.E., Alhazzani, W., Levy, M.M., et al.: Surviving sepsis campaign: International guidelines for management of sepsis and septic shock: 2016. *Intensive care medicine* **43**(3), 304–377 (2017)
15. Sakizloglou, L., Ghahremani, S., Barkowsky, M., Giese, H.: A scalable querying scheme for memory-efficient runtime models with history. *MoDELS* pp. 175–186. (2020) ACM/IEEE

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.