"The Event Model" for Situation Awareness

Opher Etzion[1], Fabiana Fournier[2], and Barbara von Halle[3]

[1]Information Systems Department, Yezreel Valley College, Israel, opher.etzion@gmail.com
[2] IBM Research – Haifa, Haifa University Campus, Haifa 3498825, Israel, fabiana@il.ibm.com
[3] Sapiens International Corporation , Barbara.vonHalle@sapiens.com

Abstract

The Event Model (TEM) is a novel computation-independent model targeted at helping non-programmers to define and manage the logic of event-driven applications. The model design is based on a collection of building blocks that comprise a set of diagrams and normalized tables to define the event business logic of an application, a set of principles that define the set of assertions that a correct model should satisfy, and a glossary to express all the business concepts. The validity of the TEM model created is checked and guaranteed through a related set of integrity principles, and automatically translated to execution by the code generator. In this paper we concentrate on the model itself. The concepts and facilities of the model are demonstrated through an example taken from the Cold Chain Management (CCM) domain. Preliminary tests in the scope of transport and logistics indicate that the tables and diagrams in TEM are well accepted and embraced by non-technical people, who stress the ease and friendly manner of defining the event logic as the main benefit of TEM.

Keywords: Event-driven applications, model driven engineering, computational independent model, conceptual modeling, real-time business intelligence.

1 Introduction

In this paper we present The Event Model (TEM), a novel way to model, develop, validate, maintain, and implement event-driven applications. The Event Model follows the Model Driven Engineering approach [1, 3] and can be classified as a CIM (Computation-Independent Model), providing independence in the physical data representation and implementation details, omitting details that are obvious to the designer. This model can be directly translated to an execution model (PSM-Platform-Specific Model in the Model Driven Architecture terminology) through an intermediate generic representation (PIM-Platform-Independent Model).

TEM is based on a set of well-defined principles and building blocks, and does not require substantial programming skills, therefore targets non-technical people. In this paper we bring a high overview of TEM and focus on the main building blocks that constitute a TEM model, that is TEM diagrams and logic tables. In TEM, the event derivation logic is expressed through a collection of normalized tables. These tables can be automatically validated and transformed into code. This idea has already been successfully proven in the domain of business rules by The Decision Model (TDM) [20]. The Decision Model groups the rules into natural logical groups to create a structure that makes the model relatively simple to understand, communicate, and manage.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

We illustrate the model throughout this paper using a scenario from the Cold Chain Management (CCM) domain. The example employed is a simplified version, yet representative, of a real-word use case in that domain. A *cold chain* is defined as a temperature controlled supply chain. One of the major issues in this field is the transportation of temperature sensitive products through thermal and refrigerated packaging methods and the logistical planning to protect the quality of these shipments. Examples of cold chain products are fruits and vegetables, pharmaceuticals, and technology products. The cold chain serves the function of keeping food fresh for extended periods and eliminating doubts over the quality of the food products. Unfortunately, about 25% of all food products transported in the cold chain are wasted each year due to breaches in integrity that cause fluctuations in temperature and product degradation[1]. *In our scenario, John Cool is the quality control officer at NeverRotten Ltd. He is in charge of setting control policies for the online monitoring of the company's cold chain products. John's task is to detect a potentially dangerous condition of a container before an actual product quality disqualification takes place, thus remediation actions can be taken, saving time and money. To this end, John wants to define two main policy rules:*

- *Alert me when, inside a container, the temperature is in the permitted range constantly increases for the last 5 minutes.*
- *Alert me whenever a delay longer than permitted occurs.*

We show how TEM can help John Cool to easily create the logic needed to monitor any delays and temperature changes in a cold chain container to achieve the goals stated above.

2 TEM in a Nutshell

This section provides a high level view of The Event Model. We discuss its origins, design goals, building blocks, and basic concepts.

2.1 TEM and Concept Computing

TEM follows the paradigm of concept computing [6], according to which all model artifacts are concepts. A *concept* is a meaningful term within the user's domain of discourse. The model consists of concepts and *semantic relationships* among concepts. These concepts are based on the user's cognitive terms, and are independent of the IT terms or implementation. The vision is to strive for automatic transformation along with model-driven engineering; this approach contrasts with the current state of practice in which the transformations between the three levels of models are mostly done manually. The vision is to have a concept-oriented model and transform it in a mostly automated fashion to create an execution model. Concept computing belongs to the family of executable specifications, which has been studied in different domains [1]. While the concept computing vision aims at simplification, the model still needs to be expressive enough to allow this automatic transformation. The success of such a model in the event-driven domain depends on the level of simplification relative to existing event-driven models. In the construction of TEM we employed some simplification goals, as discussed below.

2.2 TEM Simplification Goals

After observing and experiencing the relative complexity of event processing tools, we wanted to define simplification goals for the design of TEM so it can used by non-IT experts. In this section we outline these simplification goals.

[1] http://people.hofstra.edu/geotrans/eng/ch5en/appl5en/ch5a5en. html

1. **Stick to the basics** by eliminating technical details. Looking at designs and implementations of event-driven applications, we observe two types of logic: the *application logic*, which directly states how derived events are generated and how the values of their attributes are assigned, and *supporting logic,* which is intended to enrich events or query databases as part of the processing. In our CCM example, the temperature range can be reported as part of an event and is either produced by the sensor or enriched later by an external database. Alternatively, it may not be part of an event but rather a result of a query executed during the evaluation of a pattern from either a database or a global variable store. The first simplification design goal is to view the concept of "temperature range" as a concept that is **obvious** in the designer's terminology and thus eliminate the supporting logic of where its value resides and how it should be fetched; we move that aspect "behind the scenes". These details can be inferred automatically during the code generation phase.

2. **Employ top down, goal-oriented design**. Many design tools require logical completeness (such as referential integrity) at all times. This requires building the model in a bottom-up fashion; namely, all the meta-data elements must be defined (events, attributes, data elements) before using them in the logic definition. Our second simplification design goal is to support top down design, and allow temporary inconsistency. We allow work in the "forgive" mode [9], in which some details may be completed at a later phase. This design goal complements the "stick to the basics" goal, by concentrating on the business logic first, and completing the data aspects later.

3. **Reduce the number of logical artifacts**. In a typical event processing application, there may be multiple logical artifacts, including event processing agents, queries, or processing elements, depending on the programming model that specifies the derivation logic of a single derived event. This variety arises when there are multiple ways to create a single derived event. In our CCM example there might be different circumstances in which a delay is detected. Our design goal is to have a single logic artifact for every derived event that accumulates all the ways to derive this event. This goal reduces the number of logical artifacts and bounds it by the number of derived events. It also eases the verifiability of the system, since possible logical contradictions are resolved by the semantics of this single logical artifact.

4. **Use fact types as first class citizens in the model**. In many of the conceptual models that are descendants of the Entity Relationship model [12], terms are modeled as attributes that are subordinates of entities or relationships. In some cases, it is more intuitive to view these concepts as "fact types" and make them first class citizens of the model, so the entity or event they are associated with is secondary (and may be a matter of implementation decisions). This requirement is again consistent with the "stick to the basics" goal.

2.3 TEM building blocks

TEM is composed of two main building blocks that relate to the model itself and are the main focus of this paper. These are the diagrams (Section 3) and the logical concepts (Section 4). Additional building blocks of the model are:

- TEM Glossary: The concept dictionary used for the interpretation of a specific application.
- Integrity principles: The principles that govern the model integrity.
- Code generator: The automatic translator of a model to executable code. The code generator is able to infer information that is not explicitly stated in the model, according to the stick to the basics principle.

3 TEM Diagrams

One way to simplify the model is to apply a top-down methodology that provides a high level logical view and understanding of the system at hand.

A TEM diagram illustrates the structure of the logic by showing a situation along with the flow direction of derivations in a top-down manner. At the top of the diagram there is a goal, which is the situation that is required to be derived. This goal is connected with the raw and derived events that are identified as participants in the situation derivation. This representation is done in a recursive way until raw events or facts are encountered , as depicted in Figure 2 for our CCM example.

A TEM diagram includes nine icons that express all the relevant terms (Figure 1).

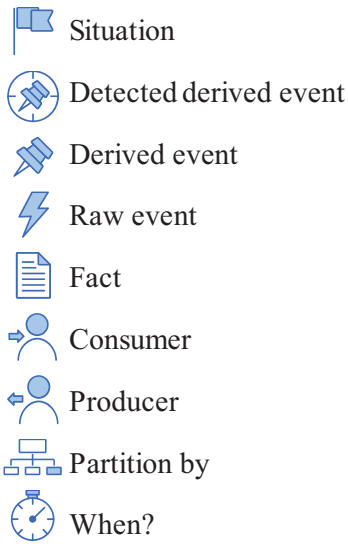

Figure 1: Product quality deterioration logic EDT

Each block in the diagram (a set of rectangle shapes, separated by connecting lines) represents a specific piece of logic with a single corresponding *Event Derivation Table* as explained in Section 4.1. The red rectangles in the background of each block represent the context for the block. The contexts can be collapsed or expanded. Dotted lines specify event flows to and from the event-driven system.

Figure 2 depicts the TEM diagram for the *Product quality deterioration* situation in our CCM example. The situation to be derived is a potential risk to the product quality, which requires alert notification and possible intervention. We have one consumer of the situation (*Quality control officer*, who gets the system alerts) and two producers: *Sensors* that emit the Sensor input; and *Shipment operations system*, which emits *Shipment starts* and *Shipment planned* raw events. The *Context* part of the *Shipment delay* derived event is expanded in the diagram to show a temporal context that is initiated when a shipment starts and ends at the *shipment planned time*, incremented by a *delay tolerance*. The *delay tolerance* indicates a grace period that is calibrated according to the specific situation. Sometimes a delay of a minute can be considered a problem, while in other cases, only a delay of a few days from the planned time is considered a situation that requires an action. We partition the events according to the *Shipment ID* domain fact type since we are looking for delays at the level of the shipment ID. Domain Fact Types serve as abstract fact types to enable segmentation contexts.

For each situation in TEM, there is a corresponding TEM diagram. The diagrams serve as a major design tool that provides a top down view. All blocks that describe situations or derived events require the definition of logical concepts.

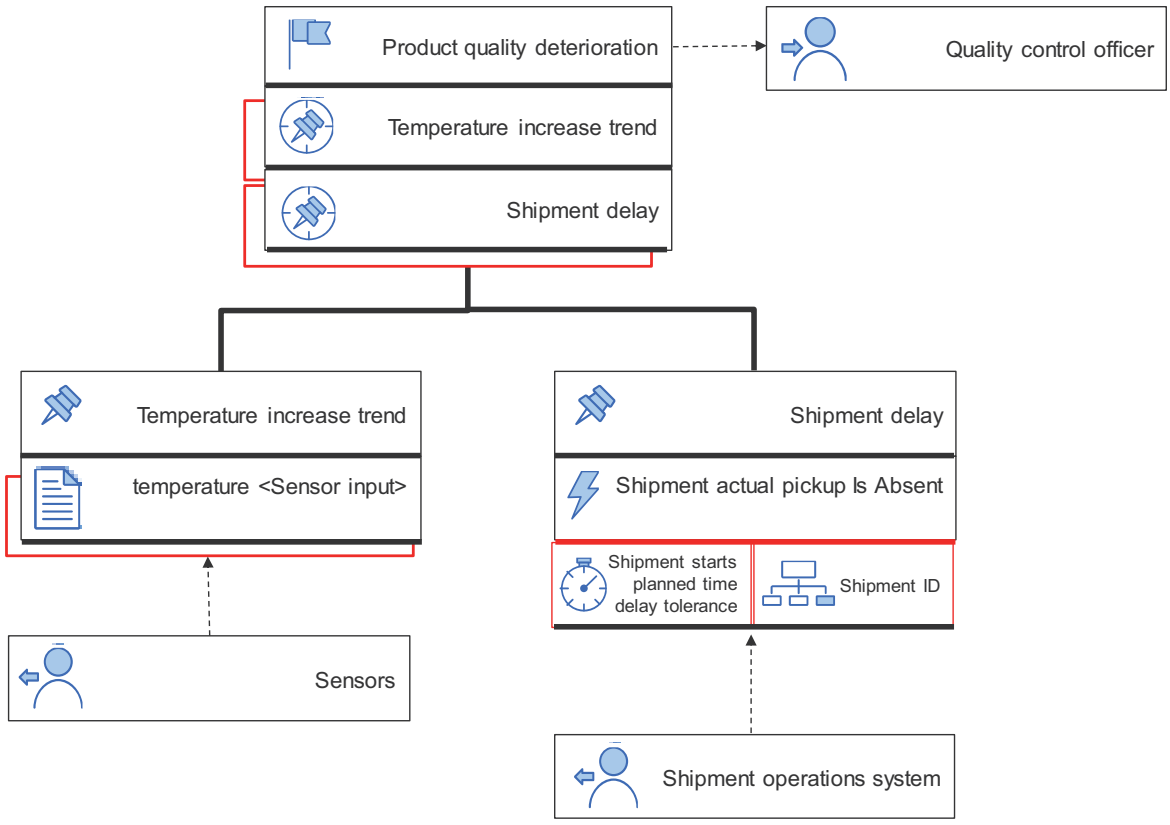

Figure 2: Temperature increase trend logic EDT

4 TEM Logical Concepts

Logical concepts are descriptions of concepts that are computed by the described application. The Event Model Logic consists of two logical concept types which are represented as tables.

Event Derivation: A single logical artifact for each derived event. The derived event mentioned in the name is associated with the table in the sense that the table specifies the conditions for generation of new instances of this event type.

Computation logic: A logical artifact that specifies the computation of assignments of the values of fact types (attributes) associated with a derived event. The derived fact type mentioned in the name is associated with the table in the sense it describes the value assignment for its fact types. Note that if the value of a derived fact type can be implicitly inferred, then the computation table for this derived fact type can be omitted.

Although the names of concepts in TEM can be determined freely by the system designer, we use some naming conventions in the logic tables for the sake of clarity. For example, domain fact types as well as event types start with a capital letter; fact types start with a lowercase letter. We also underline event types in condition columns that have an Event Derivation Table of their own (hyperlinks), to stress the fact that these events are themselves derived from another piece of logic, and enabling users to follow paths of inference by clicking these links.

We describe TEM logic tables in more detail in the following sections.

4.1 TEM Event Derivation Tables

An *Event Derivation Table (EDT)* is a two-dimensional representation of logic leading to a derived event, based on events and facts. Thus, an EDT designates the circumstances under which a derived event of interest is

reached. In our CCM scenario there are three EDTs shown in Table 1, Table 2, and Table 3 that correspond to the same names in the TEM diagram.

4.1.1 Event Derivation Tables Structure

The first row in an EDT indicates its name. The EDT name is the *derived event name* + "Logic", for example, *Product quality deterioration* Logic in Table 1. The table consists of two parts, context and conditions, separated by a red line. The context part consists of two logical sections. The temporal context, represented by *When expression, When start,* and *When end* columns; and the segmentation context represented by the *Partition by* column. For example, Table 2 describes a non-overlapping sliding fixed interval temporal context [10] of 5 minutes' length and a segmentation context that partitions the events by Container ID domain.

Table 1: Product quality deterioration logic EDT

Product quality deterioration **Logic**												
Row #	**When Expression**	**When Start**	**When End**	**Partition by**	**Filter on event**		**Pattern**				**Filter on pattern**	
				Shipment ID			*Temperature increase trend*		*Shipment Delay*			
1	*always*			*same*			*is*	*Detected*				
2	*always*			*same*					*is*	*Detected*		

Table 2: Temperature increase trend logic EDT

Temperature increase trend **Logic**										
Row #	**When Expression**	**When Start**	**When End**	**Partition by**	**Filter on event**		**Pattern**		**Filter on pattern**	
				Container ID	*temperature*		*temperature*			
1	*for every 5 minutes*			*same*	*is between*	*lower bound, upper bound*	*is*	*Increasing*		

Table 3: Shipment delay logic EDT

Shipment delay **Logic**									
Row #	**When Expression**	**When Start**	**When End**	**Partition by**	**Filter on event**	**Pattern**			**Filter on pattern**
				Shipment ID		*Shipment actual pickup*			
1		*Shipment starts*	*planned time + delay tolerance*	*same*		*is*	*Absent*		

4.1.2 Event Derivation Tables Conditions

The conditions part consists of three types of conditions. The conditions are logically applied in the following order.

Filter conditions are expressions evaluated against the content of a single event instance. The role of filter conditions is to determine whether an event instance satisfies the filtering condition and should participate in the derivation. For example, the *Filter on event* column in

Table 2 describes a condition on a fact type *temperature*, which belongs to the *Sensor input* event type. The temperature value must be between predefined bounds in a certain range.

Pattern conditions are expressions on related event types' instances such as Detected, Absent, Thresholds over Aggregations, or Fact Type value changes [10]. The role of pattern conditions is to detect the specified relationships among event instances. For example, in Table 3, the Pattern condition describes an absence detection of event type *Shipment actual pickup,* which means that no event instance of that event type is detected within the specified context.

Filter on pattern conditions are expressions on multiple event occurrences, including comparisons, memberships, and time relationships. The role of the filter on patterns conditions is to filter the pattern result based on conditions among the different events that participates in this pattern. Following the CCM example, let us assume the following scenario: we want to identify whether a shipment was picked up more than two hours after the planned time. We name this derived event *Significant shipment delay*. In this case, the pattern is *Shipment actual pickup* ***occurs after*** *Shipment planned pickup*. The filter on the pattern condition will be expressed as the difference between *shipment planned pickup time* and the *shipment actual pickup time* is greater than two hours (see Table 4).

Table 4: Example of a filter on pattern conditions

Significant shipment delay **Logic**											
Row #	**When Expres**	**When Start**	**When End**	**Partition by**	**Filter on event**		**Pattern**		**Filter on pattern**	
				Shipment ID			*Shipment actual pickup*		*occurrence time of Shipment actual pickup*	
1	*always*			*same*			*occurs after*	*Shipment planned pickup*	*is greater than*	*planned time+2*

The three types of conditions are optional, meaning they can either appear or not in an EDT, however a TEM model is valid if it contains at least one condition. We also do not restrict the number of conditions per condition type. For example in Table 2, we can add a new condition to the Pattern which specifies that in addition to checking whether the temperature value is increasing, we also check that we have at least three Sensor input events in the same Context.

The EDTs have disjunctive normal form (DNF) semantics. Each row in the table indicates a different set of circumstances in which the same event can be derived; therefore, the derived event logic is the union of the rows (logical OR relationship). On the other hand, in each row all conditions in the columns must be satisfied, therefore the columns satisfy an AND logical relationships. For example, as described in Table 1, the *Product quality deterioration* event can be derived when either a *Temperature increase trend* event is detected or a *Shipment delay* event is detected.

TEM connection is a dependency among EDTs when the conclusion, i.e., derived event, of one EDT is referenced in another EDT. Connections are shown in the TEM tables as underlines or hyperlinks. For example, *Temperature increase trend* and *Shipment delay* events are underlined in Table 1 since they are conclusions of *Temperature increase trend logic* and *Shipment delay logic EDTs*, respectively.

4.2 TEM Computation Tables

A derived event, like any event in TEM, is a container that contains facts (attributes) which are instances of the fact types contained in the derived event's event type. Part of the derivation is the assignment of values to these facts. Some of the computed facts are mere copies of values. Thus, according to the simplification goal of **stick to the basics**, their computation details may be omitted and their computation assignment is implicit. A *Computation Table* is a two-dimensional representation of logic leading to a *computed fact type* that needs to be explicitly specified. Let's assume that the *Shipment delay* derived event type has two associated fact types: *Shipment ID* and *Delay message*. The value of *Shipment ID* is computed in an obvious way, namely, by copying the value of the specific partition argument. The *Delay message* has to be explicitly computed, as shown in Table 5. Likewise, Table 6 shows the computation of the two possible alert messages associated with the *Product quality deterioration* situation (see explanation below). Note that the "+" sign denotes string concatenation.

Table 5: Delay message computation table

delay message **Computation**		
Row #		**Row in Event derivation Table**
1	*"Shipment " + Shipment ID+ " pickup time is delayed in " +delay tolerance+ "minutes "*	*1*

Table 6: Alert message computation table

alert message **Computation**		
Row #		**Row in Event derivation Table**
1	*"the temperature in container" + Container ID + "constantly increases within the last 5 minutes"*	*1*
2	*delay message*	*2*

4.2.1 Structure of Computation Tables

The first row in a computation table indicates its name, composed of the *fact type name* + "Computation". For example, Table 6 is a computation table that describes the logic to compute the *alert message* fact type associated with the *Product quality deterioration* event type. The second row is the headings row. The third row and on, include the row number, the expression value of the computed fact type, and a reference to the row number in the corresponding EDT.

Looking at *Product quality deterioration* EDT in Table 1, there are two cases in which the *Product quality deterioration* event type can be derived. One is Shipment delay and the other is Temperature trend increase. Each case dictates a different value to the computed fact type *alert message*. Table 6 contains the two possible values that can be assigned. The first row refers to the case in which a *Temperature increase trend* occurred, since the "row in event derivation table" *Shipment Delay* equals 1.

There is only one case in which the *Shipment delay* event type can be derived as shown in Table 3. In this case, the alert includes the delay elapsed time as computed in Table 5.

While the logic artifacts may be defined first, the glossary concepts eventually need to be completed at a later phase, prior to the model's validation

5 Related Work

In this section we briefly survey work related to TEM in several areas: event processing modeling, semantic modeling of events, and executable specifications.

In the area of event processing modeling, Cugola and Margara provide [5] a comprehensive survey and comparison of models, including aspects of the functional model, processing model, deployment model, interaction model, data model, time model, and rule model. In general, the event processing models contain "programming in the large" modeling, which is typically an event flow model [10] or stream processing model [8]. The "programming in the small" model is closely related programming models such as stream modeling [15] and rule based modeling [2]. Some of the modeling languages employ visualization (i.e. of the event flows) [16]. Another branch of event modeling is based on logic programming. Models in this area follow Kowalski's event calculus model [14].

The main novelties of TEM relative to existing event models are mainly two. First, it is targeted to non-technical people. This is enabled by applying a top-down approach that satisfies the simplification goals and supporting the creation of a specification without providing technical and "obvious" details, such as location of data-items. Second, TEM provides direct path to automatic implementation. This is a departure from current event models that are closely related to the implementation scheme.

The area of semantic data models [17] deals with the semantics of data and relationships among data elements. Most models follow the entity-relationship approach (ER) and its descendent methods (EER). Fidalgo et al., present a recent work [12] in which entities and relationships are first class citizens and attributes are secondary. Fact models [18] take business concepts as first class citizens, and data as containers for these facts. Our model follows the fact modeling approach, which has not been investigated yet in the area of event modeling.

The idea of executable specification was introduced in the early days of software engineering, for example by Urban et al. [19]. TEM can be considered an instance of this concept.

The Decision Model (TDM) [20] is an instance of a model that has similar goals in a different domain (decision management). The main difference between TDM and TEM is that TDM models the inference of computed values of facts as a function of other facts, while TEM models the logic of derivation of events in an event-driven context-based fashion.

6 Conclusions and Future Work

This paper presents The Event Model (TEM). TEM is a novel way to develop and implement event-driven applications. The friendly, yet rigorous, representation of the event logic enables the model to be simpler relative to existing models and accessible to people lacking IT skills. We illustrated the main logic concepts and artifacts of TEM using an example from the CCM domain. Experiments conducted in the scope of transport and logistics indicate that the tables and diagrams in TEM are well accepted and embraced by non-technical people, who stress the ease and friendly manner of defining the event logic as the main benefit of TEM. We believe that these preliminary tests are a good indicator of TEM's potential to open a new era for the consumption and pervasiveness of event-driven applications. In order to prove this statement, further experimentation is required including different domain areas and more complex scenarios.

The simplification design goals stated at the beginning of this paper have been realized as summarized in Table 7.

There are several model extensions, which are either progress or planned:

1. Support for current missing functionality, such as spatial patterns and contexts, pattern policies, and temporal correctness guards.

Table 7: Realization of simplification design goals

	Simplification goal	Realized by
1	Stick to the basics by eliminating technical details	The derivation and computation logic does not contain any logic of data fetching. This is either inferred or completed at a later phase. Assignments of values to attributes of derived events, whose assignment is obvious since they are copied from the context data, can be inferred by the system and does not have to be explicitly defined as part of the logic.
2	Employ top down, goal oriented design	The methodology supports top down, goal-oriented design by making the goal-oriented diagram a starting point. The logic tables are built in "forgive" mode, enabling reference to glossary artifacts prior to their definition.
3	Reduce the quantity of logic artifacts	The normalization principle, according to which there is a single EDT for each derived event, bounds the number of logic artifacts.
4	Use fact types as first class citizens in the model	Fact type is the fundamental basic unit in the model.

2. Support for non-functional requirements: The idea is to extend TEM to model non-functional requirements. Note that there have been some studies of high level modeling of non-functional requirements [4].

3. Extend the model to tangent activities: modeling the process of instrumentation and modeling goals for optimization based decisions.

4. Extend the model to support artifact based business state-oriented processing [13].

In addition, we are carrying out more work in model validation using constraint satisfaction techniques [7], and in code generation for various languages.

7 Acknowledgments

Fabiana Fournier has received funding from the European Union's Seventh Framework Programme FP7/2007-2013 under grant agreement 619491 (FERARI).

References

[1] Bodenstein C., Lohse F., and Zimmermann A. 2010. *Executable Specifications for Model-Based Development of Automotive Software*. SMC 2010, 727-732.

[2] Bragaglia S., Chesani F., Mello P., and Sottara D. 2012. *A Rule-Based Calculus and Processing of Complex Events*. RuleML 2012, 151-166.

[3] Brambilla M., Cabot J., and Wimmer M. 2012. *Model Driven Software Engineering in Practice*. Morgan & Claypool.

[4] Chung L and Leite C.J.P. 209. *On Non-Functional Requirements in Software Engineering*. Conceptual Modeling: Foundations and Applications (2009), 363-379.

[5] Cugola G., and Margara A. 2012. *Processing flows of information: From data stream to complex event processing*. ACM Comput. Surv. (CSUR) 44(3).

[6] Davis M. 2012. *Concept Computing: Bringing Activity-Context Aware Work & Play Spaces into the mainstream*. Keynote presentation from the Association for the Advancement of Arti?cial Intelligence 2012 conference (AAAI 12). URL: http://www.slideshare.net/Mills/understanding-concept-computing

[7] Dechter R. 2003. *Constraint Processing*. Elsevier.

[8] Dindar N, Tatbul N., Miller R.J., Haas L.M., and Botan I. 2013. *Modeling the execution semantics of stream processing engines with SECRET*. VLDB J. (VLDB) 22(4), 421-446.

[9] Etzion O. 1993. *Flexible consistency modes for active databases applications*. Inf. Syst. (IS) 18(6), 391-404.

[10] Etzion O. and Niblet P. 2010. *Event processing in action*. Manning.

[11] Farahbod R., Gervasi V., and Glässer U. 2014. *Executable formal specifications of complex distributed systems with Core ASM*. Sci. Comput. Program. (SCP) 79, 23-38.

[12] Fidalgo R., Alves E., España S., Castro, and Pastor J.O. 2013. *Metamodeling the Enhanced Entity-Relationship Model*. JIDM 4(3), 406-420.

[13] Heath F., Boaz D., Gupta M., Vaculín R., Sun Y., Limonad L., and Hull R. (2013) *Barcelona: A Design and Runtime Environment for Declarative Artifact-Centric BPM*. ICSOC 2103, 705-709.

[14] Kowalski R.A. 1991. *Logic Programing in Artificial Intelligence*. IJCAI (1991), 596-604.

[15] Jacques-Silva G., Kalbarczyk Z., Gedik B., Andrade H., Wu K-L., and Iyer R.K. 2011. *Modeling stream processing applications for dependability evaluation*. DSN 2011, 430-441.

[16] Marquardt N., Gross T., Sheelagh M., Carpendale T., and Greenberg S. 2010. *Revealing the invisible: visualizing the location and event flow of distributed physical devices*. Tangible and Embedded Interaction, 41-48.

[17] Peckham J. and Maryanski F.J. 1988. *Semantic Data Models*. ACM Comput. Surv. (CSUR) 20(3), 153-189.

[18] Ross R.G. 2000. *What Are Fact Models and Why Do You Need Them* (Part 1). Business Rules Journal, 1(5) URL: http://www.BRCommunity.com/a2000/b008a.html.

[19] Urban S.D., Urban J.E and Dominick W.D. 1985. *Utilizing an Executable Specification Language for an Information System*. IEEE Trans. Software Eng. (TSE) 11(7), 598-605.

[20] Von Halle, B., and Goldberg L. 2010. The Decision Model. CRC Press.