META: Middleware for Events, Transactions, and Analytics

M. Arnold
D. Grove
B. Herta
M. Hind
M. Hirzel
A. Iyengar
L. Mandel
V. A. Saraswat
A. Shinnar
J. Siméon
M. Takeuchi
O. Tardieu
W. Zhang

Businesses that receive events in the form of messages and react to them quickly can take advantage of opportunities and avoid risks as they occur. Since quick reactions are important, event processing middleware is a core technology in many businesses. However, the need to act quickly must be balanced against the need to act profitably, and the best action often depends on more context than just the latest event. Unfortunately, the context is often too large to analyze in the time allotted to processing an event. Instead, out-of-band analytics can train an analytical model, against which an event can be quickly scored. We built middleware that combines transactional event processing with analytics, using a data store to bridge between the two. Since the integration happens in the middleware, solution developers need not integrate technologies for events and analytics by hand. At the surface, our Middleware for Events, Transactions, and Analytics (META) offers a unified rule-based programming model. Internally, META uses the X10 distributed programming language. A core technical challenge involved ensuring that the solutions are highly available on unreliable commodity hardware, and continuously available through updates. This paper describes the programming model of META, its architecture, and its distributed runtime system.

Introduction

Business event processing consists of receiving events from external sources, evaluating some business rules, and taking an action. Automating this task frees human operators to focus on difficult cases. Event processing should be fast. When event processing scales to high-velocity event streams, it can provide value to the business by assessing many small risks and opportunities. In contrast, analytics often consist of scanning the contents of a data store to build an analytical model, using technologies ranging from relational queries to machine-learning algorithms. When analytics scale to large data sets, the analytics can provide value to the business by informing high-quality decisions. This paper is about *META* (Middleware for Events, Transactions, and Analytics), which combines the two technologies via a shared data store. In other words, with META, event processing uses the results of analytics to make better decisions.

Motivation

The following three scenarios motivate META: (1) Marketer Alerts, (2) Disease Tracking, and (3) Shopfront Ads. All three scenarios follow the high-level picture in **Figure 1**. At the center is an event processor, which receives input events, performs transactions with respect to entities in a store, uses (*scores* against) analytics results, and produces output events (also known as *actions*). Below are descriptions of the scenarios:

1. *Marketer-Alerts scenario*—Consider a business that receives an input event each time a client reads material on a product. Each client is represented by an entity in the store, along with a list of the most recent events. The analytics training consists of computing the average length of time spent reading. Given a new input event,

Digital Object Identifier: 10.1147/JRD.2016.2527419

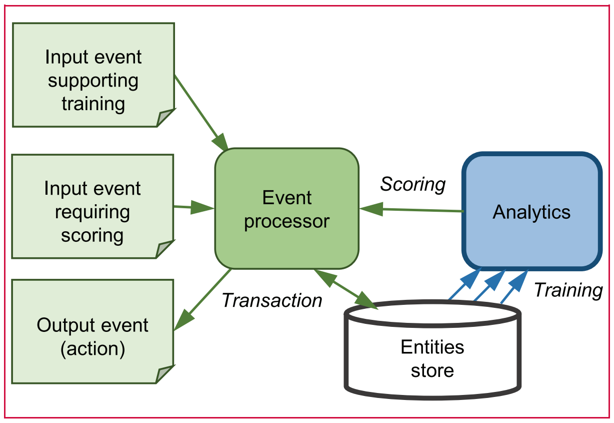

Figure 1

High-level overview of META (Middleware for Events, Transactions, and Analytics).

the event processor can score it with respect to the analytics results to check whether a client dwells unusually long on certain material. If so, it can generate an output event to alert the marketer responsible for that client.

2. *Disease-Tracking scenario*—In this scenario, input events report cases of people falling ill. The analytics training consists of finding clusters of such reports with similar symptoms, times, and locations. Then, a doctor faced with a new case can generate an input event asking for similar cases (scoring). If the new case belongs to a cluster of known cases, the output event can report back information about that cluster.
3. *Shopfront-Ads scenario*—Consider a situation in which image recognition technology, given an image of a shopper, returns a shopper description including attire, age, gender, etc. A retail shop with cameras installed at the check-out register can record pairs of ⟨`description`, `purchase`⟩ in the store. By treating the description as features and the purchase as a label, a machine-learning algorithm can train an analytical model that predicts the purchase based on the description. Cameras that watch potential shoppers walking by the shopfront can generate events with shopper description, which the event processor can score against the analytical model to predict a possible purchase. Then, the output event (action) is to display an advertisement for the predicted item to purchase on the shopfront. Note that for privacy reasons this scenarios uses no identifying information (such as a credit card number or a mobile phone number).

It is possible to implement these scenarios by using separate software applications for processing events and for performing analytics. However, doing so is difficult, as it requires understanding, installing, programming, and operating two separate complex systems. Furthermore, it is insufficient to run the analytics once, since the analytics results grow stale when new data arrives. One approach would be to periodically copy the data from the event processor to the analytics system, while transforming it to adjust between the data models of the two systems. Unfortunately, this incurs an extra performance cost, which means that it can only happen infrequently, e.g., once a night. Instead, META is an integrated platform that avoids these problems.

Note that, although the three scenarios use a diverse set of analytics, they all conform to the same general structure, shown in Figure 1. All three cases use a closed-loop system, where events produce inputs for analytics and save them in a store, and the analytics output scores for events. Once deployed, no human intervention is required to upload new analytical models. Whereas Scenario (1) uses descriptive analytics in the form of straightforward queries that compute an exact answer, Scenarios (2) and (3) use predictive analytics in the form of machine-learning algorithms that produce approximate results. Among the two predictive analytics, the clustering employed by Scenario (2) is unsupervised since it requires no labels, whereas Scenario (3) employs supervised learning that relies upon ⟨`features`, `label`⟩ pairs.

High-level overview

META is a project at IBM Research that is closely affiliated with an IBM product called "Operational Decision Manager: Decision Server Insights" [1], or *ODM Insights* for short. This paper describes META, not ODM Insights. This paper omits or simplifies some aspects of ODM Insights, while also adding other aspects that exist only as a research prototype. This paper does not make any statements about the capabilities of the product. Consult the product documentation for authoritative information about ODM Insights [1].

Incoming *events* from external systems into META are typed objects that can be represented naturally as XML (Extensible Markup Language) or JSON (JavaScript** Object Notation) documents. Events are handled by event-processing *agents*. Each agent subscribes to certain event types, and META routes events to subscribing agents. Event processing is reactive: when an event arrives, agents are activated, run some business rules, and then become passive again waiting for the next event. Each agent activation reads or writes zero or more *entities*, which are typed objects that reside in a transactional store. An agent activation manipulates a relatively small amount of data. All side effects of an agent activation, which comprise optionally modifying the entities store and emitting zero or more output events, belong to a *transaction*, which happens exactly once and is atomic. Outgoing events go either to external systems or are consumed by other agents. (An agent can subscribe to any kind of events, whether received from the outside world or produced by other agents. This is useful for building modular applications.) However, when META consumes its own events, it triggers a separate transaction.

Analytics *training* is decoupled from event processing. Training reads a large portion of the data in the store, but does not write any entities. Training runs either incrementally or on a frequent periodic schedule (for example every 15 minutes). The result of training is an *analytical model*, which may be slightly stale. Analytics *scoring* is part of event processing. Scoring happens during agent activation and consults the most recent available analytical model. We use the terms training and scoring in a general sense that encompasses both descriptive and predictive analytics. For instance, scoring may simply compare a value to a computed average, or it may traverse a decision tree.

Another way to view META is as a middleware for building *Systems of Insight*. It resides at the intersection between Systems of Engagement, which are the front-ends through which an enterprise interacts with individual consumers, and Systems of Record, which are the back-ends where an enterprise holds the authoritative copy of its data. By connecting these two technologies, META is a platform for building solutions that bring more insights to bear when engaging individual consumers.

The main challenges in the design of META involved offering a unified programming model and a scalable execution platform for events and analytics. At design time, the developer should be able to specify event rules and analytics in a uniform way. META supports this by a family of domain-specific languages for defining the data model and the code (rules and analytics). At runtime, the system should be able to handle both high-velocity event streams and high-volume analytics. META supports this by a resilient distributed architecture with a shared in-memory store. For scalable analytics, META internally uses the X10 distributed programming language [2], which was developed in-house and combines scale-out characteristics with a high-level, general-purpose developer experience.

Related work

While META integrates event processing with analytics in a single system, one can also combine them by using separate systems. In fact, the databases community has long separated OLTP (Online Transaction Processing) for reacting to high-velocity data in motion from OLAP (Online Analytical Processing) for analyzing high-volume data at rest [3]. Applying OLAP on the contents of an OLTP system traditionally requires an ETL (Extract Transform Load) step, for instance, once a night. Recent technological advances make integrated systems feasible: main memory has grown cheaper and thus larger, and distributed main-memory systems have become more scalable and resilient. Integrating the technologies in a single system leads to better performance, usability, and resilience.

HyPer is a database system that supports both OLTP and OLAP with high performance [4]. Similarly, HANA is a database system for high-performance OLTP and OLAP [5]. Both HyPer and HANA adapt relational database techniques for the case where most tables reside in memory for good performance. Whereas HyPer and HANA focus on basic database transactions and queries, META offers a higher-level programming model for business event processing and diverse analytics. Spark Streaming is an offshoot of a batch analytics platform that emulates stream processing via micro-batches [6]. Conversely, Flink** is a streaming system that also offers batch processing [7]. Naiad is a dataflow platform that supports both streaming and iterative high-volume analytics in the same application [8]. Whereas Spark Streaming, Flink, and Naiad focus on sequences of data transformation stages, META offers a higher level of abstraction with an agent-centric event processing middleware integrated with out-of-band analytics.

CQL (Continuous Query Language) is a streaming language designed around the duality between streams and relations [9]. While its notion of relations could help integrate streaming with analytics on data at rest, CQL did not further explore this design point. SPL (Streams Processing Language) is a streaming language for distributed processing using in-memory state [10]. While many SPL applications also score streams with respect to analytical models, those analytical models are trained on separate systems. ActiveSheets** uses spreadsheets for stream processing, but does not pursue large-scale analytics of data at rest [11]. Percolator is a continuous distributed incremental analytics system aiming at keeping analytical models fresh [12]. The Percolator paper focuses on the analytics, and does not explain how they can be integrated with event processing. For more on stream and event processing see the survey paper by Cugola and Margara [13].

Paper organization

The remainder of this paper is structured into four sections. The programming-model section describes the developer experience for programming solutions on top of the META middleware. The architecture-overview section examines the system internals. The section on X10 for analytics dives deeper into research aspects of the implementation. Finally, the conclusion section

Listing 1 Example code illustrating the domain-specific languages of META. Identifiers for user-defined entities and events, along with their properties, agents, and queries, are typeset in bold.

```
 1 --- data model ---
 2 a Client is a business entity identified by a name.
 3 a Client is related to a marketer (a Marketer).
 4
 5 a Read Event is a business event time-stamped by a date.
 6 a Read Event is related to a client (a Client).
 7 a Read Event has a topic.
 8 a Read Event has a length (a number).
 9
10 --- agent descriptor ---
11 'ClientRules' is an agent related to a Client,
12 processing events:
13 - Read Event, where this Client comes from the client of this Read Event
14
15 --- event-condition-action rule ---
16 when a Read Event occurs
17 if
18    the length of this Read Event is more than 'Average Read Event Length'
19 then
20    emit a new Alert where
21        the client is 'the Client',
22        the topic is the topic of this Read Event,
23        the marketer is the marketer of 'the Client';
25
26 --- global event query ---
27 define 'Average Read Event Length' as
28   the average length of all Read Events
29   during the last period of 6 hours.
```

summarizes the paper and gives an overview of ongoing research activities.

Design time: Programming model

In this section, we describe the interface that META provides to developers for programming solutions. We make the concepts and capabilities of META more concrete while leaving a discussion of how META itself is implemented to subsequent sections. The META programming model revolves around three basic concepts: entity, event, and agent. An entity is an object in the store, an event is an object on the wire, and an agent hosts code to process an event bound to an entity. This section uses the Marketer-Alerts scenario from the introduction as a running example. **Listing 1** shows the code for this scenario.

As Listing 1 shows, META provides natural-language syntax to define types, event processing rules, and analytics. This syntax does not permit free-flow natural language, but rather, consists of domain-specific languages (DSLs). Type definitions such as `Client` and `Read Event` in Listing 1 extend the vocabulary of the base DSLs, making it possible to verbalize properties of objects in rules (i.e., express the properties in a syntax approaching natural English language). While code for META is typically written by programmers, it is designed to be readable by non-programmers. This enables domain experts in the line of business to understand and give feedback on the business rules. The technology for DSLs used by META comes from the IBM Operational Decision Manager (ODM) product suite. Some ODM products even support DSLs resembling natural languages aside from English.

Specifying the data model

Lines 1 through 8 of Listing 1 define two types. In META, a *concept* is a user-defined type for an object with properties. A *business entity type* such as `Client` in Listing 1 is a concept with a mandatory *identified by* property, which holds the primary key of the object. A *business event type* such as `Read Event` in Listing 1 is a concept with a mandatory *timestamped by* property. Entities are mutable and events are immutable. Properties that are defined via *has* contain either primitive values or nested instances of other concepts. In contrast, properties that are defined via *is related to* contain the key of another entity, not its value. One feature that is not illustrated by Listing 1 involves properties that can also be defined using the plural voice to define lists.

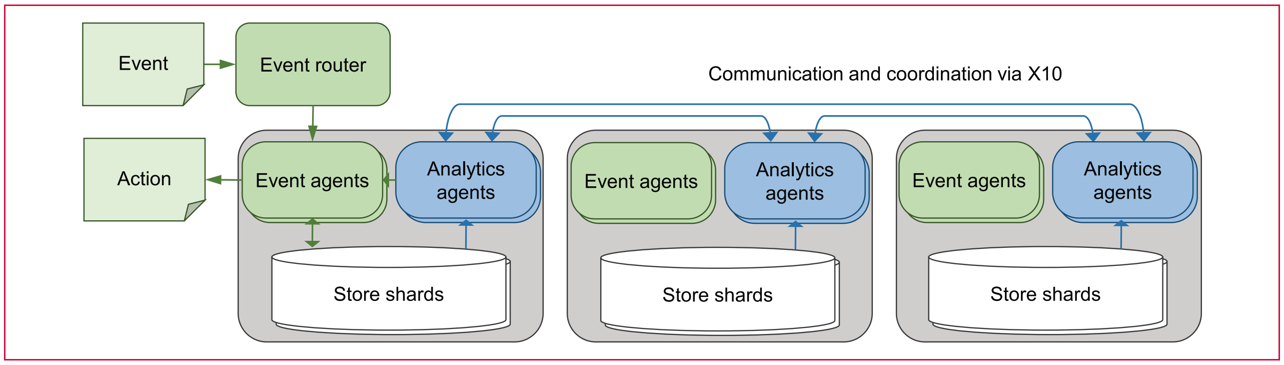

Figure 2

Distributed architecture of META.

Specifying event agents

Lines 10 through 23 of Listing 1 define an agent. An *agent descriptor* (Lines 11 through 13) subscribes the agent to events and defines how to determine the key of the bound entity given an event. META supports three kinds of event-handling agents: rule agents, Java** agents, and predictive scoring agents. The agent in Listing 1 is an example of a rule agent. A rule agent defines event-condition-action rules using the *when-if-then* syntax illustrated in Lines 16–23. Each rule has implicit variables in scope for the triggering event (here, *this* `Read Event`) and the bound entity (here, *the* `Client`). Aside from the surface syntax, the code resembles production rules such as OPS5 (Official Production System 5), where the condition can involve pattern matching, and the action can involve modifying or creating new working memory elements [14]. Some features that are not illustrated by Listing 1 are rules that can also include aggregations, temporal predicates, and geo-spatial expressions. One benefit of the rule-based approach is that it is possible to add new rules without disrupting the rest of the system. This enables a more modular and agile development process.

Specifying analytics

Lines 26 through 29 of Listing 1 define analytics. The example illustrates a *global event query*, which ranges over all events of a certain type, i.e., not just the events bound to a particular entity. Another kind of analytics in META not illustrated here is the *global entity query*, which ranges over all entities of a certain type. Event queries and entity queries perform descriptive analytics over data in motion and data at rest, respectively. In relational terminology, they typically involve selection, projection, and aggregation. Aside from these forms of analytics, META also supports predictive analytics via SPSS* [15]. SPSS supports a large variety of algorithms, including, but not limited to, various decision trees, various clustering algorithms, logistic regression, neural networks, anomaly detection algorithms, Bayesian networks, and support-vector machines. For predictive analytics, the training happens in SPSS, and the scoring happens in a predictive scoring agent (one of the three kinds of event agents mentioned above).

Overall, META offers a consolidated programming model for both events and analytics. This facilitates authoring solutions that combine these two technologies in order for the insights from analytics to inform the actions of event processing.

Runtime: Architecture overview

Now that we have seen how a developer can implement a solution on top of META, we examine the internals to see how META itself is implemented. Recall from the introduction that the three central components of META are the event processor, the store, and the analytics.

One of the design objectives of META is that it should scale to large volumes of events and entities by using the memory and compute resources of a cluster of computers, not just a single computer. Therefore, META is designed as a distributed system, using several processes that each have event agents, store shards, and analytics agents, as depicted in **Figure 2**. Taken together, the store shards constitute a partitioned in-memory store. A *partitioning function* maps from an object key to a shard and the process that own it. Since event agents are bound to entities, and entities are partitioned across all shards, the code (for instance, the rules) for all event agents exist in all processes.

The green path in Figure 2 illustrates event processing in META. Consider an input event, e.g., a Read Event `{date:"2015-0729-10:17:37.231",client: "Chuck",topic:"tech",length:"23:28.310"}`.

An *event router* consults the partitioning function to determine the destination process. For the above event, that would be the process containing the entity for Chuck, e.g., `{name:"Chuck", marketer:"Mary"}`. That process activates the subscribing event agent, which performs a transaction against the local shard that contains the bound entity. The activation may also involve scoring against an analytical model and generating output events, e.g., sending an alert to Chuck's marketer, Mary. The green path is optimized for low latency and high throughput by using a relatively small amount of local independent in-memory data per activation.

The blue path in Figure 2 illustrates analytics in META. Analytics agents in all processes scan data from all store shards in parallel. The analytics agents perform data-parallel local computation, punctuated by global communication and coordination with each other. The result of analytics is an analytical model, which can subsequently be used for scoring during event processing. The blue path is optimized for volume by using data parallelism to analyze a substantial portion of the global in-memory data.

Another design objective of META is that it should be resilient. First, META is designed for *high availability*: it survives node failure without downtime or data loss. To achieve high availability, META uses a replicated store, where each object has a primary copy and a secondary copy in different shards on different computers. Second, META is designed for *continuous availability*: it supports software upgrades without downtime. To achieve continuous availability, META uses rolling upgrades. It takes advantage of the high-availability support by stopping processes with old code one at a time, and starting processes with new code to replace them. Third, META is designed for *disaster recovery*: even if all nodes fail, it can restart without data loss. META enables disaster recovery via write-through to a durable backup store: each transaction saves essential data to disk. This is an optional feature that can be enabled or disabled depending on solution requirements.

At the level of the programming model described in the previous section, solution developers need not concern themselves with how the system is implemented internally. However, in the interest of advancing the state of the art by sharing our experience with building META, here we mention some technology choices. The store is WebSphere* eXtreme Scale (WXS). The processes use the WebSphere Application Server Liberty Core on a Java Virtual Machine (JVM**). META uses the OSGi (Open Services Gateway initiative) module system for Java, which Liberty implements. The rule agents are implemented by compiled Rete engines [16]. And the analytics are implemented on the basis of Java and X10, as outlined in the following section. Note that since these choices are not exposed to the solution developer, they may change. Furthermore, technology choices in META do not necessarily reflect those in ODM Insights.

In this section, we described in broad strokes how META is implemented by a resilient distributed system. On a three-server cluster, with high-availability enabled but disaster-recovery disabled, META achieves throughputs above 10,000 events per second with subsecond latencies. The following section dives more deeply into one of the challenging implementation aspects of META.

X10 for analytics

Recall that in META, entity state is distributed across a cluster of machines. Performing analytics over this state requires orchestrating calculations and data movements over the cluster. The X10 programming language and runtime system are designed to ease the development and deployment of such codes. META uses X10 to implement analytics. In this section, we provide a brief introduction to X10 and discuss the use and integration of X10 in META.

X10 and APGAS

The X10 programming language [2] is an imperative, object-oriented language built on the APGAS programming model (Asynchronous Partitioned Global Address Space) [17]. In this model, a distributed system is viewed as a collection of places. An X10 application runs over a collection of places, possibly large, possibly heterogeneous. X10 makes it easy to construct aggregate objects and orchestrate aggregate tasks that span over multiple places. For example, the X10 standard library offers distributed arrays implemented in X10 [18]. These arrays support distributed operations such as reductions. A typical X10 program first implements and constructs a series of distributed data structures, then computes over a global view of the data, e.g., sequences of map-reduce jobs over distributed key-value maps. More information on X10 can be found online [19].

X10 as a Service

The X10 programming concepts closely align with the META concepts of distributed entities and global queries, making it natural to express META analytics in X10. Moreover, the X10 compiler can generate Java code from X10 code [20, 21], providing language compatibility with the rest of META—META is primarily implemented in Java and runs across a cluster of JVMs. Nevertheless, running X10 analytics and the X10 runtime as a service in a META cluster requires solving key technical challenges, which we now discuss. A similar and possibly even more significant effort would be needed to embed other distributed analytics frameworks (such as Spark [6]) as a subsystem of a well-behaved piece of middleware.

X10 was initially developed for the high-performance computing (HPC) audience to run scientific applications on the IBM Power* 775 and other high-performance, high-availability systems [22]. As a result, the X10 runtime was biased toward launching and running a single application with high performance on dedicated resources assumed to be reliable. In contrast, in META, X10 is intended to support running 1) a series of jobs, 2) using a persistent pool of Java Virtual Machines managed by META, 3) shared with other META services 4) running on commodity clusters. These clusters can experience node failures, live addition of nodes, or rolling upgrades.

We review these four requirements in turn:

1. To run a series of jobs using a common X10 runtime instance, we augment the X10 runtime with APIs (application programming interfaces) for startup and shutdown, job submission, and monitoring. Moreover, we refactor the runtime to eliminate the global state and replace it with per-job state, making it possible to run multiple jobs in parallel without interference.
2. To run X10 as part of a META cluster, we developed new X10 runtime mechanisms for launching, linking together, and destroying X10 places within the JVMs of META. Instead of launching new X10 processes directly with our own launcher, we bootstrap the startup of the runtime as an OSGi service, and use the META data storage layer to communicate initial bootstrap information about the runtimes. This mechanism enables us to establish connections among X10 places following the topology of the META cluster, and gives X10 programs direct and fast access to the data stored within the META JVMs. We expand on our bootstrap implementation later in this document.
3. To properly share system resources with other META components, we make sure idle threads in the X10 runtime do not waste CPU time. In the HPC context, polling or busy waiting minimizes latency using CPU time that is wasted anyway, as the system is dedicated to running a single program at any point in time. In the META context, idle threads are suspended so that other META services can use the CPU if needed, or CPUs can be throttled to reduce energy consumption.
4. To support live cluster reconfigurations—node loss, addition, or upgrade—we proceeded in steps. In a first release of META, we encapsulated the X10 runtime in a reloadable bundle, with an X10 management service monitoring the cluster and X10 state. This made it possible to unload, reconfigure, and reload the X10 runtime upon cluster reconfigurations, on demand. In a second release, we added the ability to reconfigure the X10 runtime as it is running. In particular, we make it possible to establish and close connections between X10 places at any time. When a cluster change is detected, connections to missing places are closed, new places are started on JVMs joining the cluster, and new connections are established. While the X10 runtime itself now survives cluster reconfigurations, we still cancel all in-progress analytic jobs since the jobs are not capable of adjusting dynamically to new cluster configurations. Our cancellation implementation guarantees that cancelled jobs cannot interfere with new jobs. The implementation reports cancelled jobs so that these jobs can be resubmitted to the X10 service, if needed.

To facilitate building and deploying X10 in the context of META, we also replace our native communication library with a pure Java implementation over TCP/IP sockets, thereby eliminating all native code and JNI (Java Native Interface) wrappers from the X10 runtime.

Our bootstrap code deserves some additional discussion. As mentioned above, we do not launch X10 ourselves, but rather embed the startup of X10 within the startup of the rest of META, so that the cluster itself is managed by META scripts and utilities, without concern for X10 directly. Our implementation consists of two OSGi bundles; additionally, we make use of two unique data map types in WebSphere eXtreme Scale (WXS), described below, which are independent from the normal maps used to hold META data. Our first OSGi bundle is a small X10 management bundle, which performs the launching function for the second OSGi bundle, which is the X10 runtime itself. There are additional bundles which are the X10 programs for computing global aggregates. They depend on the X10 runtime bundle and share the same lifecycle. We configure a per-JVM map (a CONTAINER_MAP in WXS terms) that does not hold any actual data, but whose configuration we can query at runtime to identify the number of JVMs in the cluster and thus the number of places we should expect in X10. Our X10 management bundle partially initializes the X10 runtime bundle in each container, which opens up a listen socket at some dynamic operating-system provided port number. The management bundle then gathers up the port number of the local X10 runtime, combines it with the hostname defined for the WXS container, and stores this information into a single-partition WXS map, accessible from any JVM in META. This co-locates the hostname and port information for every place into a single JVM, from which we can then retrieve, sort, and compare the number of entries to the expected count from the per-JVM map. Once our management bundle in each JVM determines that it has the connection information for all other places, it configures the X10 runtime bundle to initialize the communication links to other places, and begin running jobs. There is additional logic to handle JVM additions or removals in all phases of startup and after running jobs. The per-JVM maps continue to be monitored for place addition and removal. The single-partition map is consulted whenever a new JVM comes online, to detect if X10 is already running in the cluster, and to get the associated hostnames and ports for the new place to connect to.

Conclusion

In this paper, we discussed META—Middleware for Events, Transactions, and Analytics. META made it possible for business event processing to use insights learned from the context to arrive at the best actions. We provided an overview of the META programming model. By offering a unified programming model for both event processing and analytics, META simplified solution development. Next, we provided an overview of the in-memory distributed architecture of META. This architecture was designed to scale while also offering a high level of resilience. Finally, we provided a deep-dive on how the X10 programming language was used to implement parts of META. X10 is a general-purpose distributed programming language that makes it easier to implement distributed analytics than doing so from scratch in a non-distributed language such as Java. Overall, META is a middleware for writing systems of insight that combine events with analytics.

Much additional research on META is possible. We note that the text in the following paragraphs is not a statement about the future roadmap of the ODM Insights product, which is ultimately driven by customer requirements.

We are exploring how to use rule languages for more general descriptive analytics. CAMP (Calculus for Aggregating Matching Patterns) lays a foundation for this [23]. We are also exploring more diverse predictive analytics implemented in X10, including parallel DBSCAN (Density Based Spatial Clustering of Applications with Noise) [24]. More generally, on the front of predictive analytics, we are investigating tighter integration of META with SPSS [15], using our M3R (Main-Memory Map Reduce) technology [25] as a common foundation. One fundamental trade-off for predictive analytics is between the frequency of retraining, which incurs a performance cost, and the accuracy of the analytical model, which tends to degrade when training is less frequent. To address this issue, we are developing AQuA (Adaptive-Quality Analytics).

We are exploring how to optimize event processing performance [26]. We have demonstrated that META can be integrated with InfoSphere* Streams [10], the high-performance stream processing platform of IBM. The integration uses Streams as a front-end to consume a "firehose" of raw events and turn it into a smaller volume of higher-value business events for META. One major performance factor for META concerns the transactional store; other isolation levels could yield better performance [27]. For certain event handling patterns in META, there are also known techniques with "good" algorithmic complexity based on sliding-window aggregation [28] and finite-state machines [29]. (We use the phrase "good algorithmic complexity" to indicate at most $\log(N)$ time per event, where N is the amount of state maintained by the system.) We are exploring how to use those to simplify and optimize stateful rule agents. Finally, we are investigating cloud-hosting for META. One aspect of this that we are exploring is using a NoSQL or SQL JSON store for META [30].

Finally, we are exploring how to add fault tolerance and elasticity to the X10 programming language [31] to enable applications such as META analytics to detect place failures or cluster reconfigurations, and seamlessly recover and adapt to the new configuration. Overall, META has a diverse research agenda with an active technology transfer pipeline.

Acknowledgments

We thank the ODM Insights team (especially Steve Demuth, Stephane Mery, Dan Selman, and David Martin) for a fruitful collaboration. We also thank the anonymous reviewers for constructive and meticulous feedback.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Oracle America, Apache Software Foundation, or Component Software, Inc., in the United States, other countries, or both.

References

1. IBM Corporation, *Operational Decision Manager: Decision Server Insights*. [Online]. Available: http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1
2. P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, C. von Praun, V. Saraswat, and V. Sarkar, "X10: An object-oriented approach to non-uniform clustered computing," in *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2005, pp. 519–538.
3. S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," in *International Conference on Management of Data (SIGMOD)*, 1997, pp. 65–74.
4. A. Kemper and T. Neumann, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *International Conference on Data Engineering (ICDE)*, 2011, pp. 195–206.
5. V. Sikka, F. Farber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhovd, "Efficient transaction processing in SAP HANA Database: The end of a column store myth," in *Demonstration at the International Conference on Management of Data (SIGMOD-Demo)*, 2012, pp. 731–742.

6. M. Zaharia, T. Das, H. Li,T. Hunter, S. Shenker, and I. Stoica, “Discretized streams: Fault-tolerant streaming computation at scale,” in *Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 423–438.
7. Apache, *Flink: Scalable Batch and Stream Data Processing*. [Online]. Available: https://flink.apache.org
8. D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, “Naiad: A timely dataflow system,” in *Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 439–455.
9. A. Arasu, S. Babu, and J. Widom, “The CQL continuous query language: Semantic foundations and query execution,” *Journal on Very Large Data Bases (VLDB J.)*, vol. 15, no. 2, pp. 121–142, Jun. 2006.
10. M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soulé, and K.-L. Wu, “IBM streams processing language: Analyzing big data in motion,” *IBM J. Res. & Dev.*, vol. 57, no. 3/4, paper 7, pp. 7:1–7:11, 2013.
11. M. Vaziri, O. Tardieu, R. Rabbah, P. Suter, and M. Hirzel, “Stream processing with a spreadsheet,” in *European Conference on Object-Oriented Programming (ECOOP)*, 2014, pp. 360–384.
12. D. Peng and F. Dabek, “Large-scale Incremental Processing Using Distributed Transactions and Notifications,” in *Operating Systems Design and Implementation (OSDI)*, 2010, pp. 251–264.
13. G. Cugola and A. Margara, “Processing flows of information: From data stream to complex event processing,” *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, Jun. 2012.
14. C. L. Forgy, “OPS5 user’s Manual,” Carnegie Mellon University (CMU), Tech. Rep. 2397, 1981.
15. IBM, *SPSS software: Predictive Analytics Software and Solutions*. [Online]. Available: http://www.ibm.com/software/analytics/spss/
16. C. L. Forgy, “Rete: A fast algorithm for the many pattern/many object pattern match problem,” *Artificial Intelligence*, vol. 19, no. 1, pp. 17–37, Sep. 1982.
17. V. Saraswat, G. Almasi, G. Bikshandi, C. Cascaval, D. Cunningham, D. Grove, S. Kodali, I. Peshansky, and O. Tardieu, “The asynchronous partitioned global address space model,” in *Workshop on Advances in Message Passing (AMP)*, 2010.
18. D. Grove, J. Milthorpe, and O. Tardieu, “Supporting array programming in X10,” in *Workshop on Libraries, Languages and Compilers for Array Programming (ARRAY)*, 2014.
19. X10 Open-Source Community, *X10: Performance and Productivity at Scale*. [Online]. Available: http://x10-lang.org
20. M. Takeuchi, Y. Makino, K. Kawachiya, H. Horii, T. Suzumura, T. Suganuma, and T. Onodera, “Compiling X10 to java,” in *X10 Workshop (X10)*, 2011, pp. 3:1–3:10.
21. M. Takeuchi, D. Cunningham,D. Grove, and V. Saraswat, “Java interoperability in managed X10,” in *X10 Workshop (X10)*, 2013, pp. 39–46.
22. O. Tardieu, B. Herta, D. Cunningham, D. Grove, P. Kambadur, V. Saraswat, A. Shinnar, M. Takeuchi, and M. Vaziri, “X10 and APGAS at Petascale,” in *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2014, pp. 53–66.
23. A. Shinnar, J. Siméon, and M. Hirzel, “A pattern calculus for rule languages: Expressiveness, compilation, and mechanization,” in *European Conference on Object-Oriented Programming (ECOOP)*, 2015, pp. 542–567.
24. M. Patwary, M. Ali, D. Palsetia,A. Agrawal, W. Liao, F. Manne, and A. Choudhary, “A new scalable parallel DBSCAN algorithm using the disjoint-set data structure,” in *Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2012, pp. 1–11.
25. A. Shinnar, D. Cunningham, B. Herta, and V. A. Saraswat, “M3R: Increased performance for in-memory Hadoop jobs,” in *Conference on Very Large Data Bases (VLDB) Industrial Track*, 2012, pp. 1736–1747.
26. M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, “A catalog of stream processing optimizations,” *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, Apr. 2014.
27. H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O’Neil, and P. O’Neil, “A critique of ANSI SQL isolation levels,” in *International Conference on Management of Data (SIGMOD)*, 1995, pp. 1–10.
28. K. Tangwongsan, M. Hirzel, S. Schneider, and K.-L. Wu, “General incremental sliding-window aggregation,” in *Conference on Very Large Data Bases (VLDB)*, 2015, pp. 702–713.
29. M. Hirzel, “Partition and compose: Parallel complex event processing,” in *Conference on Distributed Event-Based Systems (DEBS)*, 2012, pp. 191–200.
30. M. Enoki, J. Simeon, H. Horii, and M. Hirzel, “Event processing over a distributed JSON store: Design and performance,” in *Conference on Web Information System Engineering (WISE)*, 2014, pp. 395–404.
31. D. Cunningham, D. Grove, B. Herta, A. Iyengar, K. Kawachiya, H. Murata, V. A. Saraswat, M. Takeuchi, O. Tardieu, “Resilient X10: efficient failure-aware programming,” in *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2014, pp. 67–80.

Received August 5, 2015; accepted for publication August 28, 2015

Matthew Arnold *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (marnold@us.ibm.com).* Dr. Arnold is a Research Staff Member at IBM and manages the Cloud Continuous Quality group. His research interests involve developing analytics and tooling to aid program understanding, optimization, testing, and debugging of modern cloud applications.

David Grove *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (groved@us.ibm.com).* Dr. Grove is a Principal Research Staff Member at IBM and a Fellow of the Association for Computing Machinery. He currently co-leads the X10 research project. His research interests include the analysis and optimization of object-oriented languages, virtual machine design and implementation, scalable runtime systems, Just-In-Time compilation, online feedback-directed optimization, and automatic memory management.

Benjamin Herta *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (bherta@us.ibm.com).* Mr. Herta focuses on distributed programs, both in the realm of High Performance Computing (HPC), and in application frameworks such as Apache Spark. For META, Mr. Herta was primarily focused on the networking code used by the X10 runtime, and integration of X10 with the META data store and job execution.

Michael Hind *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (hindm@us.ibm.com).* Dr. Hind is a Distinguished Research Staff Member and Senior Manager of the Programming Technologies Department. Dr. Hind is an ACM Distinguished Scientist and Associate Editor of *ACM Transactions on Architecture and Code Optimization*. His research interests include programming models and their implementations, static and dynamic development tools, and middleware for emerging commercial paradigms.

Martin Hirzel *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (hirzel@us.ibm.com).* Dr. Hirzel is a Research Staff Member at IBM and manages the Programming Languages research group. His research concerns programming languages, event and stream processing, and analytics. He also serves as the Architect of Analytics for the ODM Insights product.

Arun Iyengar *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (aruni@us.ibm.com)*. Dr. Iyengar performs research and development on distributed computing, cloud computing, and Web performance, as well as fault-tolerant computing and high availability at the IBM T. J. Watson Research Center.

Louis Mandel *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (lmandel@us.ibm.com)*. Dr. Mandel is a Research Staff Member at IBM. His main research topic is on the design and implementation of programming languages for reactive systems.

Vijay A. Saraswat *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (vsaraswa@us.ibm.com)*. Dr. Saraswat is currently a Distinguished Research Staff Member in Cognitive Computing, working on logic, knowledge representation, and machine learning. He is also Chief Scientist for the IBM Operational Decision Management (ODM) business unit. Previously, he started and led the META project, and was responsible for the design and initial implementation of the analytics capability for ODM Insights. He also started and co-led the X10 high performance programming language effort. His main research interests are in logic, distributed systems, programming languages, and artificial intelligence.

Avraham Shinnar *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (shinnar@us.ibm.com)*. Dr. Shinnar is a Research Staff Member at IBM Research. His research focuses on programming languages, including work on type systems, formal verification, and distributed computing.

Jérôme Siméon *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (simeon@us.ibm.com)*. Dr. Siméon is a Research Scientist at IBM Watson with a background in databases and programming languages. He was a major contributor to the field of XML (Extensible Markup Language) processing, including XQuery of WC3 (World Wide Web Consortium). His current interests include languages and compilers, as well as transaction management for NoSQL databases.

Mikio Takeuchi *IBM Research Division, Tokyo Research Laboratory (mtake@jp.ibm.com)*. Mr. Takeuchi is a Research Staff Member at IBM leading the design and implementation of Managed X10 (X10 for the Java virtual machines). His research interests include programming languages, high performance computing, distributed computing, and analytics.

Olivier Tardieu *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (tardieu@us.ibm.com)*. Dr. Tardieu is a Research Staff Member at the IBM T. J. Watson Research Center, leading the design and implementation of the X10 runtime. His research interests include parallel programming models and languages, HPC systems, software safety, and fault tolerance.

Wei Zhang *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (weiz@us.ibm.com)*. Dr. Zhang is a Research Staff Member at IBM and a member of the Programming Languages research group. His research interests include parallel programming, large-scale machine learning, and concurrent software reliability.