A Middleware for Data-centric and Dynamic Distributed Complex Event Processing for IoT Real-time Analytics in the Cloud

Gustavo B. Baptista, Felipe Carvalho, Sergio Colcher and Markus Endler

Department of Informatics – Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil.

{gbaptista,fcarvalho,colcher,endler}@inf.puc-rio.br

Abstract. *IoT big data real-time analytics systems need to effectively process and manage massive amounts of data from streams produced by distributed data sources. There are many challenges in deploying and managing processing logic at execution time in those systems, especially when 24x7 availability is required. Aiming to address those challenges, we have developed and tested a middleware for Distributed CEP, with a data-centric and dynamic design, based on the Data Distribution Service for Real-Time Systems (OMG-DDS) specification and its extension for dynamic topics/types (DDS-XTypes). Its main advantages include the use of OMG-DDS; witch is suitable for IoT applications with QoS requirements, its dynamic capabilities, and the scalable and parallel execution of the CEP rules on a dynamic set of processing nodes.*

1. Introduction

Processing data streams is one of the main challenges facing today's distributed and pervasive systems, given the high dynamicity, heterogeneity and volume of real-time data generated by contemporary network-centric systems. Emerging Internet of Things (IoT) [Greengard 2015], Big Data [Nathan and Warren 2015] and Real-time Data-Stream Analytics [Byron 2014] applications (e.g. smart cities, smart energy grids, bio-surveillance, environmental monitoring, fleet monitoring and smart manufacturing) need to effectively process and manage massive amounts of data from streams produced by distributed data sources, in order to perform real-time analytics for supporting timely decision-making and timely response. Other application domains of real-time analytics include social networks, stock market analysis, IT infrastructure monitoring, etc. Data streams are infinite flows of data generated by wide array of fixed and mobile sensors or devices used in modern pervasive/mobile IoT applications. However, there are challenges in deploying and managing processing logic for these data streams in a distributed manner on datacenters or clouds [Erl et al. 2013]. In particular, there is usually a high operational cost involved in setting up a distributed data stream processing environment or dynamically modifying its processing logic without stopping the whole system, specially for stream analysis systems demanding 24x7 availability. Such dynamic changes may become necessary to accommodate variant needs from the application domain, such as inclusion of new patterns of data/events to be detected, or technology upgrades, i.e. changes of the basic data types of the input streams due to new types of sensors. Among data stream processing techniques, Complex Event Processing (CEP) [Luckham 2001] is one of the most complete models. It views data items as notifications of events from multiple sources, and uses rules for describing patterns of low-level events, that, when detected, produce higher-level events whose occurrence are notified to the interested parties. In many CEP systems, the processing logic is described as a network of event processing agents (EPAs), or EPN. In a Distributed CEP system (DCEP), an EPN can be distributed with EPAs deployed across many different machines [Etzion and Niblett 2010]. Aiming to address the challenges of setting up and dynamically managing the deployment of mechanisms for CEP-based data stream processing, we have developed and tested a middleware for DCEP, based on the Data Distribution Service for Real-Time Systems (DDS) [OMG 2006] specification and its extension for dynamic topics/types (DDS-XTypes) [OMG 2014], to support on-line detection of patterns. Its main advantages include the use of DDS; witch is suitable for IoT applications with QoS requirements, its dynamic capabilities, and the scalable and parallel execution of the CEP rules on a dynamic set of processing nodes on a datacenter or in the cloud. The middleware uses the RTI Connext DDS [RTI 2016] as its communication backbone.

2. Data-Centric Paradigm

In distributed systems, different processes need to manage and exchange data. In the same way as with data management and persistence, communication middleware can be designed around different entities, which we refer to as different communication paradigms. This paradigm influences the design of the middleware clients (e.g. services, applications or other systems), their functional capabilities and compliance to non-functional requirements. In a *message-centric* communication paradigm, the unit of information exchange is the message where the main focus is the delivery of messages to destinations, without deep knowledge of message's structure and contents, i.e. data is sent as an opaque sequence of bytes (the payload inside messages), and serialization is delegated to the application. Many communication technologies follow this approach, ranging from simple point-to-point connections to more advanced publish/subscribe communication infrastructures, such as, Java Message Service (JMS) [Oracle 2016a], Advanced Message Queuing Protocol (AMPQ) [OASIS 2012]. Some message-centric systems, known as content-based publish/subscribe systems, such as PADRES [Fidler et al. 2005], have mechanisms that enhance the communication model to be able to access the contents of messages, allowing the routing of messages by defining expressions of interest based on event properties. Another similar paradigm is the *log-centric* approach offered by systems such as Kafka [Apache Software Foundation 2016], where data topics are managed as append-only logs, which can be partitioned among different Kafka brokers for parallelism. With a *data-centric* communication paradigm [Schneider 2012], the mean of interaction is data. The middleware infrastructure has the definition of the structure of the data it manages and it is fully aware of the contents (i.e. values) of these structures, and imposes rules on how data is structured, changed and accessed (i.e. how participant nodes receive updates). While in this model the communication is peer-to-peer, there is a notion of a *global shared data space*, which is decentralized and maintained by all peers and contains the structure and instances of data, and nodes access this space in order to read and write information. The role of the infrastructure is to ensure that all participant nodes have a consistent and up-to-date view of this data space. Two main aspects that can be influenced by those different paradigms are how distributed state is managed and how it is communicated among nodes of a distributed system. The management of distributed state is a fundamental design decision, and the communication paradigm will have a direct relation on the type of interaction between distributed nodes and with the state sharing and management approach. State management in message-centric systems has to be performed by applications since the middleware is not aware of the messages contents. Since with a data-centric approach messages are not sent about data, but data itself is kept in a global data space, state is inherently disseminated and managed (i.e. reflected into the global data space) by the infrastructure. The infrastructure has the definition of data schemas, and the instances of these schemas are kept as globally accessible objects. In the same way as in relational databases, data-centric middleware provide a fixed set of operations over the global data space, with an important and significant impact on reducing systems integration efforts [Schneider 2012; Tambe 2012]. In order to realize the data-centric paradigm, a middleware implementation following this approach is necessary. In the next section, we summarize the OMG DDS specification, which is a standard for real-time data-centric publish/subscribe middleware, and a few implementations of this specification.

2.1. Data Distribution Service for Real-Time Systems (OMG-DDS)

The Data Distribution Service for Real-Time Systems (DDS) [OMG 2006] standard defines a fully distributed peer-to-peer (i.e. broker-less) real-time data-centric publish/subscribe communication model. It provides high performance communication, scalability and availability, and supports the specification of Quality of Service (QoS) contracts between data producers and consumers, and also mechanisms for dealing with real-time aspects (e.g. priority and other specific QoS policies). It allows interoperability across different DDS implementations, programming languages and platforms, as well as automatic discovery of DDS publishers/subscribers. DDS Topics are first class entities for information transfer, which application peers can publish or subscribe to, and can be regarded as distributed relational database tables. Topics are defined by the application programmer, which writes an IDL file describing the names, data types, and the keys that identify the instances (that can be seen as database rows) of each Topic, to which updates are called *samples*. A compiler takes the IDL as input, and generates code for communication stubs in a desired programming language. By using the generated stubs, applications can join a DDS Domain and publish and subscribe to data in this Domain. The DDS Domain, which contains the global shared data space, is fully distributed over the peer-to-peer network formed by participating network nodes, without any intermediate broker or centralized management entity. DDS applications produce and consume data to this global data space without any direct knowledge of the parties involved in the production and consumption of data (i.e. naming and location transparency). The DDS specification provides a large set of QoS features and enforcement mechanisms. Every producer and consumer of data defines its QoS requirements or capabilities through QoS contracts that are matched by a protocol, called Requested-versus-Offered protocol, which checks QoS matching before any establishment of communication. Those QoS parameters address many aspects, such as communication reliability, latency or transport priority, data persistence, and behavior of node discovery mechanisms, among many others.

3. Data-Centric and Dynamic Design Approach

In DCEP systems, we will herein refer to the mechanisms responsible for disseminating data as the *communication layer* (communication middleware, data objects, marshaling, etc), and the mechanisms responsible for processing the data (e.g. event processing agents (EPAs) and event processing networks (EPNs), CEP engines, event types, etc) as the *processing layer*. There are scenarios where knowing all event types and situations (i.e. CEP rules in EPAs) to be detected beforehand is not possible. Teams responsible for monitoring activities must be able to define those events and situations on the fly, to support uninterrupted monitoring. In a dynamic scenario, the DCEP system processing and communication layers must be capable of dealing with the definition of those logical entities dynamically, and keep those layers consistent with each other. Our dynamic and data-centric approach aims at achieving a fully dynamic and integrated environment for DCEP entities and event types both at the processing and communication layers.

3.1. Global Reactive Data Space

In order to do so, in this work, the concept of the *global shared space* of the DDS specification is extended to include not only topics that implement the actual transmission of data, but also the modeling of processing entities, i.e. event types and DCEP entities, in a data-centric way. By doing so, dynamic capabilities are provided in a way that promote interoperability and reduce complexity in accessing and managing the DCEP system, while obtaining the other benefits of using DDS for the dissemination of actual events, such as shared state management and consistency, decoupling, availability, scalability and performance. We propose that in addition to the DDS Topics that perform the actual transmission of data, this data space should be used to keep the description of the entities of the DCEP system. The global data space would keep two additional mechanisms: a global dictionary of event types that can be agreed upon by all participants in a communication, and the definition of DCEP entities, such as EPAs, EPNs, CEP Rules, etc and deployment information for these entities in any desired processing node. Thus, the data dissemination and reactive behavior over that same data are modeled together inside the global data space. By extending the global data space with reactive behavior, we call it the *global reactive data space*. The dynamic data-centric solution has to be implemented by three main middleware services: one for the definition of event types, other for the generation and management of dynamic DDS topics, and finally, the processing layer of the DCEP system that uses the above services, which allow the definition of DCEP entities and deployment information for those entities. While on a static approach the mapping of event types in the processing and communication layers would had to be performed by external mechanisms, that generate and keep track of static DDS topics for each existing event type in the processing layer, with our approach, when an event type is defined, both processing and communication layers have the same description of it, and a dynamic DDS topic is created for this type automatically. In order to do so, we make use of the extension specification for Dynamic DDS Topics, called DDS-XTypes [OMG 2014], to which RTI Connext DDS [RTI 2016] provides an implementation for. In addition to all the benefits obtained from using data-centric middleware for the dissemination of events, as mentioned in the previous section, the modeling of reactive behavior inside the global shared space allows dynamic capabilities to be fully supported without the use of many external dynamic allocation mechanisms, bringing benefits in terms of increased decoupling and interoperability, reduced complexity in management, evolution and maintainability. By using this approach, any data consumer, producer or EPA, can be defined in a dynamic way, only by specifying the data type it produces or consumes, and start writing or listening for updates on that type. There is no need to define any software module implementing the data type: it is defined or retrieved from the metadata service in the global data space. This facilitates the dynamic deployment of producers, consumers and especially of DCEP entities. For example, a complete EPN implementing a processing logic could be defined at runtime, only requiring the specification of the event types to be written into the metadata service, and then the definition of EPAs and their rules referencing those types. Since the event types referenced by the CEP rules are in the same global data space, and all have correspondent dynamic DDS topics associated to them, the dynamic deployment is seamless.

4. D3CEP Middleware

D3CEP (Data-Centric Dynamic Distributed CEP) is a DCEP middleware/system, which follows a clustered distribution model. It allows the deployment of Event Processing Agents (EPAs) on distributed nodes tightly coupled in a computer network in a cluster of physical machines or cloud computing instances. Those agents are logically connected to each other to form Event Processing Networks (EPNs) that compose global processing logics to detect situations. This clustered distributed model aims to provide high throughput and low latency among processing nodes, and to provide scalability in the number of event producers, consumers, and situations to be detected [Cugola and Margara 2012].

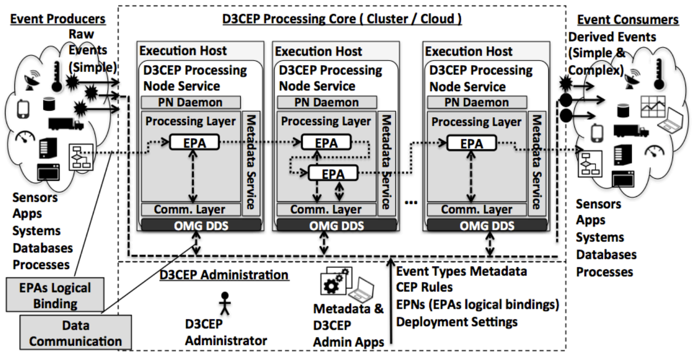

Figure 1. D3CEP Solution Overview

D3CEP uses a data-centric paradigm for the communication and the processing of events. In order to achieve data-centric communication, the D3CEP Communication Layer implements a middleware API using a Data-Centric Publish-Subscribe (DCPS) model provided by an implementation of the OMG DDS [OMG 2006] specification. The use of this peer-to-peer DCPS communication model aims to provide an optimized performance for the routing of events among processing nodes that execute EPAs. The middleware also handles the management of state among EPAs for the processing of distributed CEP rules, so EPAs, event producers and consumers don't need to perform this management. These features promote a reduction in the coupling among all entities (e.g. producers, consumers, rules, etc). Furthermore, in accordance to a data-centric approach for communication, the D3CEP Processing Layer uses a data-centric CEP engine implementation [EsperTech 2016], at each of the processing nodes, with CEP rules deployed inside EPAs, which in turn are deployed on those engines. In order to meet the requirements of dynamic and flexible behavior, D3CEP provides flexibility in the definition and re-definition of event types, rules and EPNs at execution time, without requiring stopping the system, event producers or consumers. Figure 1 depicts an example of the main elements of the solution, shown with a typical organization of a distributed complex event processing system, with event producers, which input raw events into the system, a processing core composed by a set of distributed processing nodes, where EPNs can be composed by the deployment of EPAs, and event consumers, which subscribe to receive derived events from the system. EPNs are represented in the figure by the thin dashed lines connecting all EPAs, which are logically bounded to each other, while the bold dashed lines connecting the execution hosts represent the actual data communication between the EPAs.

4.1. Middleware Layers

The main middleware layers of the D3CEP solution are the Processing Node Daemon, Metadata Sharing Layer, Processing Layer and the Communication Layer. Figure 1, presented previously, depicts an example of the middleware layers deployed on execution hosts, inside each node's PN Service instance. Figure 2 details the layers and their main internal components. The ***Processing Node Daemon*** periodically collects the node's information and publishes a heartbeat event inside the D3CEP domain, indicating the nodes current status and availability for the deployment of EPAs. This information can be consumed, for example, by an administration application (or an automatic deployment mechanism) for information useful for the deployment of EPAs.

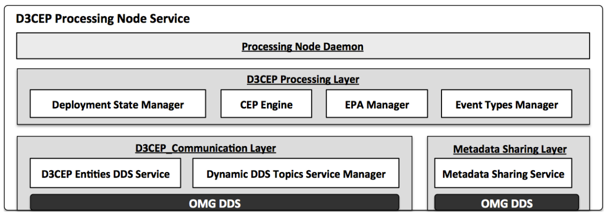

Figure 2. The D3CEP Middleware Layers.

In order to provide dynamic capabilities, the ***Metadata Sharing Layer*** provides mechanisms for the dynamic definition of metadata. The ***Metadata Sharing Service*** is a DDS-based service that allows the sharing of data types descriptions. Therefore, it provides peers in a DDS domain with a view of all existing data types descriptions in this shared data space, so that applications can query them or subscribe to receive updates. It provides an API that facilitates the development of applications that access a shared space with metadata descriptions. The ***Metadata*** element is a logical structure that describes a data type. A Metadata instance has a name and a set of members. Each member has a name and a type (e.g. string, integer, etc). For example, a metadata instance could be named as "Location", and composed by members named as "latitude" and "longitude" both of the type "double". The service is used in D3CEP as the global catalog of event types, which defines the event types to be exchanged by the Communication Layer, as to be processed by the Processing Layer. The administrator of the system can define the event types through the Metadata Administrator application, which uses the Metadata Sharing Service to publish Metadata instances into the DDS domain. When an event type is defined by the administrator, all processing nodes, event producers and consumers that have an instance of the service running on the same domain will have a view of the existing event types.

At the base of the D3CEP solution is the ***Communication Layer***, which implements data dissemination inside the D3CEP core, implementing middleware APIs on top of the OMG DDS specification. This layer is used for two main purposes: disseminating D3CEP entities, such as EPNs, EPAs, deployment settings, etc, and simple and complex events from event producers, consumers and among processing nodes. Therefore, two main services are used inside this layer, the D3CEP Entities DDS Service and the Dynamic DDS Topics Service, which are explained below. The ***D3CEP Entities DDS Service*** is a DDS service used inside the D3CEP core, which implements the sharing of entities that represent D3CEP management information, allowing an administration application to define CEP entities, such as EPNs and EPAs and their deployment. The system administrator defines the management information and publishes it into the domain. Processing nodes are passive entities, and react deploying entities locally when they are published. Therefore, there is a concept of a *Deployment State*, which is a snapshot of the system current configuration state. A Deployment State contains a set of EPNs defined, each of them composed by a set of EPAs. Also inside a Deployment State instance, there are Processing Nodes Deployments, which are a mapping of the existing EPAs to Processing Nodes. The Processing Layer on the correspondent processing nodes defined by this mapping will deploy these EPAs locally, when the Deployment State is received in those nodes. The ***Dynamic DDS Topics Service*** is a middleware API that facilitates the development of DDS applications and services that need dynamic DDS topics described at runtime. It provides a way of creating dynamic DDS topics by providing a simple data description (the aforementioned Metadata class). It is used in D3CEP to allow a data-centric and dynamic dissemination of simple and complex events from event producers, consumers and event processing agents. The CEP rules in D3CEP are dynamically defined, and dynamically deployed in distributed processing nodes to receive events as input and to produce events as output. The event types referenced in those rules need to have a mapping to underlying DDS topics for the communication to take place. However, due to this dynamic aspect, the topics cannot be defined at development time, to be statically generated. Therefore, this service is used to implement the creation of such dynamic topics, and is used by the Processing Layer for the aforementioned data dissemination. It is a generic DDS service that facilitates the creation of dynamic DDS topics, following the OMG DDS Xtypes specification [OMG 2014]. It can be used by any application or service (not only D3CEP) that needs to perform DDS communication with Dynamic DDS Topics. With this service, a client can provide a dynamic data type description (i.e. a Metadata object instance, from the Metadata Sharing Service) to create a dynamic DDS topic correspondent to this type description, in execution type, without requiring the static description and compilation, at development type, of the DDS topic through an IDL. The OMD DDS XTypes [OMG 2014] is an extension to the OMG DDS [OMG 2006] specification which allows Topics to be defined dynamically, and at execution time, without requiring the compilation of an IDL file to generate classes and support types at development time. At the time of the writing of this work, the only DDS product that implements the XTypes specification is RTI Connext DDS [RTI 2016], from Real-Time Innovations, but is claimed by PrismTech that OpenSplice DDS [PrismTech 2016] will implement the specification soon. Since XTypes is a specification of the Object Management Group (OMG), and both these companies are the main players in DDS middleware, and are participant in the elaboration of the specification, it is expected that it will become a standard feature in all DDS implementations in the near future.

The D3CEP ***Processing Layer*** implements the event processing mechanisms, which supports the dynamic definition and deployment of EPAs, so as to compose distributed EPNs. Each Processing Layer instance contains a local CEP engine, which may have several deployed EPAs. The deployed EPAs process simple and complex events received from outside the system and from other EPAs and generate derived events that are consumed by other EPAs (e.g. running locally or at other nodes) and by event consumers (i.e. subscribed monitoring applications or services). Each Processing Layer instance uses the Communication Layer for two purposes; the dissemination of D3CEP system entities that describe the configuration state of the system, such as EPNs, EPAs, deployment assignments of EPAs to processing nodes, among others; The dissemination of simple and complex events from event producers, consumers and among processing nodes. The EPAs deployed in a processing node use it to receive events as input, and to produce events as output into the DDS domain. The ***Deployment State Manager*** (DSM) manages the global deployment state of the D3CEP system. A DeploymentState is a representation of the global D3CEP system state, to which every processing node can access through the DDS global data space provided by the Communication Layer. The DSM on each processing node registers a listener into the D3CEP Entities DDS Service for receiving DeploymentState instances. When a DeploymentState is received from the network, the DSM gets the deployment information from it and verifies if there are EPAs assigned to be deployed to its local processing node (the id of the local node provided by the Processing Node Daemon). If there are EPAs to be deployed locally, the EPA reference is passed to the EPA Manager to be locally deployed. The ***EPA Manager*** manages the EPAs deployed locally in a processing node. A D3CEP_EPA has a set of rules, where each rule is composed of sequential D3CEP_Statements, which are Event Processing Language (EPL) statements describing the CEP rule processing logic. EPL was chosen since it is very similar to SQL, and is currently one of the most popular event processing languages. When the EPA Manager receives a D3CEP_EPA from the Deployment State Manager, it performs some steps to deploy the D3CEP_EPA into the local CEP Engine, and to plug the EPA input and output to the Communication Layer. After the Administrator has published a DeploymentState, and the Processing Layer in the processing nodes has performed all deployment steps, the EPAs deployed on each node are bound to Dynamic DDS Topics in the Communication Layer. The ***Event Types Manager*** performs the integration of event types at the Processing Layer with the Communication Layer, specifically for managing the mapping of event types at both levels. An Event Type instance is a representation of an event type, which contains the event type name, its Metadata description (obtained from the Metadata Sharing Service), and a correspondent Dynamic DDS Topic (obtained from the Communication Layer) for data dissemination through the DDS Domain. The D3CEP system is managed by an administrator, which is responsible for defining the description of the event types exchanged and processed by the system, CEP rules, EPAs, and EPNs, and the deployment of those entities on the distributed execution nodes of the processing core. In order to facilitate administration, two prototype applications have been developed: The Metadata Administrator and D3CEP Administrator. ***The Metadata Administrator Application*** facilitates the management of Metadata descriptions, by reading an XML file with the Metadata, and instantiating the Metadata Sharing Service and adding the Metadata to it, so it is published into the DDS domain. The ***D3CEP Administration Application*** facilitates the management of D3CEP Entities, by reading an XML file with those entities. Among the entities defined in the configuration repository are: The Deployment State to be set into the system, which contains EPNs, composed by EPAs. Each EPA has Statements, which are ordered EPL statements that implement the EPA's logic, external functions implemented in Java and called by these statements, a set of ordered execution parameters, such as arguments to rules. Also, it contains the mapping of EPAs to processing nodes. References to external relational databases can also be defined. External databases can be used as knowledge bases for rules, or else, for storing events generated by the system, for example for auditing and generation of reports.

5. Related Work

The first initiatives in establishing a data-centric model for complex event processing are the ones addressing the integration of clustered DCEP systems using the DDS protocol in a static way, to which we refer to as a *Static Data-Centric* approach, where event types and rules are defined in a static way, and DDS topics are statically generated at compile time to support the data dissemination among data producers, consumers and processing nodes. In [Corsaro 2011c; Corsaro 2011a; Corsaro 2011b], the author presents the usage of OpenSplice DDS [PrismTech 2016] to integrate centralized Esper [EsperTech 2016] CEP engines to form a DCEP system (*OpenSplice+Esper*). Using a similar approach, in [RTI 2011], the RTI Connext DDS [RTI 2016] middleware is used to integrate Oracle CEP [Oracle 2016b] engines (*RTI+OracleCEP*), and in [Oberoi 2007] RTI Connext DDS is used to integrate Coral 8 CEP engines (*RTI+Coral8*). Following a static approach, no mechanism is defined for the definition and agreement of a common representation for event types, nor any dynamic mechanisms are available for the definition of those event types and CEP rules that use those event types. As the most direct relation to this thesis are DCEP systems that implement model similar to the one proposed to our approach, to which we call *Dynamic Integrated Model* approach, meaning that the processing and communication layers are strongly integrated/coupled. *Oracle CEP* is a DCEP system with a clustered deployment model. It is a system that implements capabilities that are similar to the ones provided by our approach. The system uses for data dissemination a distributed cache solution, called Oracle Coherence [Oracle 2016c], which provides a scalable, high-performance and fault-tolerant distributed cache. A distributed cache allows event instances to be directly referenced by any participant in the communication, so data producers, consumers and processing nodes read and write event instances to this distributed cache, which seamlessly propagates those instances to all participating nodes. In order to establish a common agreement for event representation, Oracle CEP provides an event types repository. Dynamic capabilities are achieved by the usage of the OSGI [Alliance 2016] component-based framework. The system provides graphical tools for defining event types and processing entities, such as EPNs, EPAs, etc. Another system with this approach is *Solar* [Chen et al. 2008], which is a middleware for pervasive and context-aware applications. Solar allows the system allows the construction of so-called Context Fusion Networks (CFN), which has a similar purpose as the EPNs in CEP. Data dissemination is established by a scalable and self-organizing peer-to-peer overlay network based on Pastry [Rowstron and Druschel 2001], which implements a distributed hash table, which propagates data streams from data producers, processing nodes and data consumers. The dynamically integrated approach of Solar is achieved by an attribute-based representation of data, which is agreed upon by all components in the system through the use of a distributed directory service, which provides naming and discovery functionalities. Therefore, dynamic capabilities in the definition of context processing queries are possible. We also have considered message-centric systems with dynamic capabilities, to which we call *Dynamic Message-centric* systems. TIBCO

Business Events [TIBCO Business Events 2016] is a DCEP system with a networked deployment, which uses other TIBCO products, such as the TIBCO Rendezvous [TIBCO 2016] publish/subscribe middleware for data dissemination, and allows the dynamic definition of events and rules, without stopping the system.

5.1. Comparison of Related Work

There are several systems and research efforts in the area of distributed complex event processing. Scalability, naturally, is the main motivation of all distributed solutions. Regarding the integration paradigm used, a few initiatives, such as the *OpenSplice+Esper*, *RTI+OracleCEP* and *RTI+Coral8*, show how to integrate DCEP using the DDS data-centric publish/subscribe middleware, but their approaches are static, which doesn't offer the flexibility required by the dynamic requirements of some monitoring applications. Other solutions, such as *Oracle CEP* and *Solar* follow a more evolved approach with dynamic aspects and also a global representation of event types to which all participants can agree to, so that applications can be developed and deployed dynamically using the systems. These dynamic integrated solutions use Distributed Caching and Distributed Hash Table respectively as their communication technologies for data dissemination. Among the message-centric solutions, not all work provide detail about the underlying technology being used, but among them are JMS implementations, point-to-point connections, and publish/subscribe implementations. Among the investigated message-centric solutions, only TIBCO Business events provide a fully integrated and dynamic model, although requiring the usage of proprietary TIBCO products. Table 1 shows a comparison table with the related work and the investigated aspects. The D3CEP solution, presented in this thesis, is included in the table, in order to facilitate comparison. A few of the investigated solutions provide capabilities for achieving a fully integrated and dynamic model for communication and processing of events.

Related Work	Integration Paradigm	Communication Technology	Dynamic
OpenSplice+Esper	Static Data-centric	OMG-DDS (OpenSplice DDS)	No
RTI+OracleCEP	Static Data-centric	OMG-DDS (RTI Connext)	No
RTI+Coral 8	Static Data-centric	OMG-DDS (RTI Connext)	No
Oracle CEP	**Dynamic Integrated**	**Dist. Cache (Oracle Coherence)**	**Yes**
Solar	**Dynamic Integrated**	**Dist. Hash Table (Pastry)**	**Yes**
D3CEP	**Dynamic Data-centric**	**OMG-DDS + XTypes (RTI Connext)**	**Yes**
TIBCO Business Events	**Dynamic Integrated**	**Prop. pub/sub (TIBCO Rendezvous)**	**Yes**

Table 1. Comparison of related work

These solutions are marked in bold in Table 1 (including the D3CEP system). Among these solutions, only TIBCO Business Events has a message-centric design, the others use a similar design approach, but using different protocols from our solution. The communication technologies used by these systems are differentiated from our approach, since we use the DDS protocol for implementing a dynamic and data-centric design. To the best of our knowledge, a fully integrated and dynamic approach for the definition and agreement of event types and processing entities with a data-centric paradigm, and using the DDS protocol, is not yet addressed by other work. By using the DDS protocol, we consider its advantages in performance, by relying on its optimized peer-to-peer routing of data, and capabilities in meeting requirements of a wide range of applications, considering its capabilities for supporting many different levels of performance and real-time strictness, which can be explored in industrial and mission critical scenarios. By providing a rich set of QoS policies, the DDS protocol can support future capabilities in defining QoS policies at detection level (i.e. for CEP rules), which would reflect in the underlying communication layer, such as presented in [Appel et al. 2010], which describes a fully QoS aware infrastructure. Although this is not in the scope of this work, our solution provides a base framework that can be explored by future work in this aspect.

6. Use Case and Performance Evaluation

As an example use case, we define a scenario of a telemetry application, which receives data from mobile nodes (*MNs)*, such as vehicles. Those nodes periodically send a set of predefined raw events that can be used for the detection of many types of situations. For our scenario, we consider the events that contain geographical coordinates and the measurement error of these coordinates, to which we call *Location*. Therefore, with the system up and running, the Metadata Administrator application is run with an XML file defining this event type. For illustrating and evaluating performance and scalability, we then define a simple situation to be detected, but with a processing complexity that demands a distributed processing to share the load in order to handle an increasing number of MNs. Therefore, we assume a scenario of tracking MNs locations and triggering alerts when nodes are too close from one another. This situation could be used, for example, for air traffic control, situational awareness in command and control missions [T.A. DuBois et al. 2012; Baptista, Endler, et al. 2013] or for the detection of many types of mobility patterns [Baptista, Roriz, et al. 2013]. Thus, we define a complex event, called *TooClose*, which represent that a pair of MNs are within a specified distance D of each other. In order to create this event, we use the D3CEP Administration application to read an XML description defining an EPA called *TooClose, which* processes MNs Location events, combining all pairs of location events that are within distance D + 2ε from each other, where ε is the measurement accuracy. When this condition is true, the pair of Location events is aggregated into a TooClose complex event.

```
Too Close EPA
INSERT INTO TooClose
SELECT A, B
FROM Location.win:time_batch(30 sec) as A,
           Location.win:time_batch(30 sec) as B
WHERE A.nodeId <> B.nodeId AND
distance(A.latitude, A.longitude, B.latitude, B.longitude) < D + 2ε
```

Figure 3. CEP Rule of the Too Close EPA

Each EPA operates in a time-batch scheme, where it retrieves a stream window with all Location events in a given time period (e.g. all Location events from timestamp T to timestamp T - 30 seconds). The agent correlates this set of location events with the set itself, producing a product set of all possible pairs of location events. This result is then filtered by the distance function, that checks if each pair is close to each other, as shown in Figure 3. If the coordinates are within a minimum orthodromic distance (considering de accuracy of both coordinates), then the EPA produces an TooClose event containing the pair of *Location* events [Baptista, Roriz, et al. 2013]. Since the processing of the CEP rule in the ***TooClose EPA*** requires the product set of all possible pairs of Location events, in order to detect any node that is close to any other node, the considered area of interest in the map is divided into sub-regions that are considered by a single EPA, since it is only necessary to correlate the location of nodes in the same area. Therefore, we assume that each mobile node has a local function that is capable of using a location filter, defined in terms of latitude and longitude intervals, to assign an identification of a sub-region to each Location event. In this way, EPAs can only process *Location* events on the same sub-region, and many EPAs can be instantiated to support the detection of many sub-regions in a scalable manner.

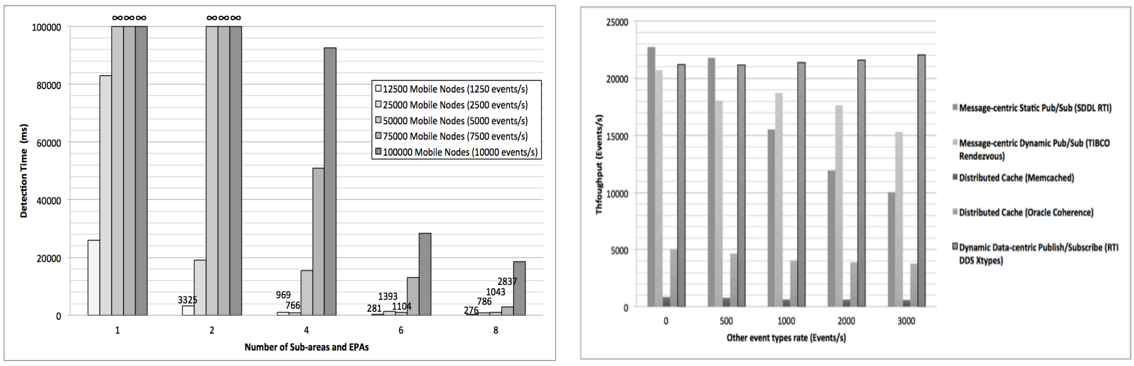

Figure 4. Detection time (a) and throughput (b) tests.

We distributed the processing among a set of processing nodes, showing how this can enable scalability of rules that have a high demand for processing. The simulation was executed in a delimited scope with sub-regions delimited by rectangles of a map, which represent the sub-region that each EPA subscribes. The application simulates 12500, 25000, 50000, 75000, 100000 MNs on this map. The number of nodes is divided among the number of instantiated EPAs (up to 8 EPAs). All those MNs send Location events that don't trigger the TooClose EPA rule, since they are on stationary locations that are distant from one another, and their goal is just to introduce load for the system to be tested. After all the nodes are sending data, a single pair of Location events that are actually close to one another is sent, with the goal of triggering the TooClose CEP rule. Since the MN Simulator is also a subscriber of TooClose events, when this event is triggered, the simulator measures the detection time of the rule. Figure 4a shows the results of the detection time test. The results reflect the scalability of the system, by increasing the number of processing nodes, the lower is the detection time. Instead of evaluating the performance of complete related DCEP systems, we have decided to isolate their communication layers, and use the same D3CEP processing layer, on top off those different communication technologies and integration models, so that the comparison is more direct. Different integration models DCEP systems that are able to support dynamic aspects were evaluated: The D3CEP system, with a **Dynamic Data-centric** model. With the ***Dynamic Message-Centric*** model: the TIBCO Rendezvous [TIBCO 2016] and Scalable Data Distribution Layer (SDDL) [David et al. 2012], which allow the dynamic definition of messages' structures. With the ***Dynamic Integrated*** model: we tested implementations of a Distributed Cache: Memcached [Fitzpatrick 2004] and Oracle Coherence [Oracle 2016c]. Therefore, we measure the throughput of the system using those different technologies, i.e. how many location instances per second are received by CEP engines. We established 3 EPAs that are capable of measuring the input rate of Location events. The MN Simulator is set to simulate 240000 mobile nodes sending events every 10 seconds, producing 24000 events per second. Three EPAs are set to receive and measure the input events rate. In addition to the Location events, the simulator produces an increasing rate of events of other types. So, we compare the sending of 0, 500, 1000, 2000 and 3000 events/s of types alternative to the Location event. Figure 4b shows the results for the throughput test. The results show the D3CEP system with the highest throughput, when the number of distinct event types increases. This is due to the fact that in a data-centric communication, in contrast to other models, the DCEP nodes don't need to open every message in order to inspect the contained event type. The data-centric middleware will only route the events with types that are in fact used by the deployed CEP rule.

7. Conclusion and Future Work

This paper presented a middleware for DCEP, with a data-centric and dynamic design approach based on the DDS specification, that supports on-line detection of patterns on cloud architectures. We have presented its architecture and its main aspects, and tests regarding performance and scalability. The use of DDS in our work provides base for future work using DDS for supporting DCEP, e.g. for IoT and mission critical applications, which may require QoS contracts at the detection level, as in [Appel et al. 2010].

8. References

Alliance, OSGI. 2016. OSGI. (2016). Retrieved April 10, 2016 from https://www.osgi.org/

Apache Software Foundation. 2016. Apache Kafka. (2016). Retrieved January 1, 2016 from http://kafka.apache.org/

Appel, Stefan, Sachs, Kai, and Buchmann, Alejandro. 2010. Quality of service in event-based systems. In *GvD Workshop*. Bad Helmstedt, Germany.

Baptista, Gustavo L.B., Endler, Markus, Dubois, Thomas A., and Johnson, Randall L. 2013. Middleware Supporting Net-Ready Applications for Enhancing Situational Awareness on Rotorcraft. In *9th International Conference on Autonomic and Autonomous Systems (ICAS 2013)*. Lisbon.

Baptista, Gustavo L.B., Roriz, Marcos, Vasconcelos, Rafael, Olivieri, Bruno, Vasconcelos, Igor, and Endler, Markus. 2013. On-line Detection of Collective Mobility Patterns through Distributed Complex Event Processing. *Monogr. em Ciência da Comput.* 13 (2013).

Byron, Ellis. 2014. *Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data*, Wiley.

Chen, Guanling, Li, Ming, and Kotz, David. 2008. Data-centric middleware for context-aware pervasive computing. *Pervasive Mob. Comput.* 4, 2 (April 2008), 216–253.

Corsaro, Angelo. 2011a. Complex Event Processing with OpenSplice DDS. *PrismTech* (2011).

Corsaro, Angelo. 2011b. Getting Started with OpenSplice and Esper. *PrismTech* (2011).

Corsaro, Angelo. 2011c. Stream Processing with DDS and CEP. (2011). Retrieved April 10, 2016 from www.prismtech.com

Cugola, Gianpaolo and Margara, Alessandro. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.* (2012).

David, L., Vasconcelos, R., Alves, L., André, R., Baptista, G., and Endler, M. 2012. A Communication Middleware for Scalable Real-time Mobile Collaboration. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*. Toulouse, 54–59.

Erl, Thomas, Puttini, Ricardo, and Mahmood, Zaigham. 2013. *Cloud Computing: Concepts, Technology & Architecture*, Pearson Education.

EsperTech. 2016. Esper. (2016). Retrieved April 10, 2016 from http://esper.codehaus.org/

Etzion, O. and Niblett, P. 2010. *Event processing in action*, Manning Publications Co.

Fidler, E., Jacobsen, H.A., Li, G., and Mankovski, S. 2005. The PADRES Distributed Publish/Subscribe System. In *8th International Conference on Feature Interactions in Telecommunications and Software Systems*.

Fitzpatrick, Brad. 2004. Distributed caching with memcached. *Linux J.* (2004).

Greengard, Samuel. 2015. *The Internet of Things*, MIT Press.

Luckham, D.C. 2001. *The power of events: an introduction to complex event processing in distributed enterprise systems*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Nathan, Marz and Warren, James. 2015. *Big Data - Principles and best practices of scalable realtime data systems*,

OASIS. 2012. OASIS Advanced Message Queuing Protocol. , October (2012), 0–124.

Oberoi, Supreet. 2007. Introduction to Complex Event Processing & Data Streams. , August (2007), 20–23.

OMG. 2006. Data-Distribution Service for Real-Time Systems (DDS). (2006).

OMG. 2014. XTypes: Extensible and Dynamic Topic Types for DDS.November (2014).

Oracle. 2016a. Java Message Service. (2016). Retrieved April 10, 2016 from http://www.oracle.com/technetwork/java/jms/index.html

Oracle. 2016b. Oracle CEP. (2016). Retrieved April 10, 2016 from http://www.oracle.com/technetwork/middleware/complex-event-processing

Oracle. 2016c. Oracle Coherence. (2016). Retrieved April 10, 2016 from http://www.oracle.com/technetwork/middleware/coherence/overview/index.html

PrismTech. 2016. OpenSplice DDS. (2016). Retrieved April 10, 2016 from www.prismtech.com

Rowstron, Antony and Druschel, Peter. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middlew. 2001* , November 2001 (2001), 329–350.

RTI. 2011. Integrating RTI Data Distribution Service Applications with Oracle CEP. *Real-Time Innov.* (2011), 1–9.

RTI. 2016. RTI Connext DDS. (2016). Retrieved April 10, 2016 from www.rti.com

Schneider, Stan. 2012. What ' s the Difference between Message Centric and Data Centric Middleware ? *Electron. Des.* (2012).

DuBois, Thomas, Blanton, Brendan, Reetz III, Ferdinand, Endler, Markus, Kinahan, William, Baptista, Gustavo, Johnson, Randall. 2012. Open Networking Technologies for the Integration of Net-Ready Applications on Rotorcraft. Annual Conference American Helicopter Society (2012).

Tambe, Sumant. 2012. Communication Patterns Using Data-Centric Publish-Subscribe. In *Silicon Valley Code Camp*.

TIBCO. 2016. TIBCO Rendezvous. (2016). Retrieved April 10, 2016 from http://www.tibco.com/products/automation/enterprise-messaging/rendezvous

TIBCO Business Events. 2016. TIBCO Business Events. (2016). Retrieved April 10, 2016 from http://www.tibco.com/products/event-processing/complex-event-processing/businessevents