CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Conference Paper

Extending publish/subscribe mechanisms to SOA applications

Michele Albano
Luis Lino Ferreira
José Sousa

CISTER-TR-160402

2016/05/03

Extending publish/subscribe mechanisms to SOA applications

Michele Albano, Luis Lino Ferreira, José Sousa

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: mialb@isep.ipp.pt, llf@isep.ipp.pt, 1110852@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The Event Handler systems is part of theArrowhead Framework, which aims to apply Service OrientedArchitecture to the embedded systems' world. The EventHandler system is a component that supports the handling ofevents, and in that sense it enriches service-oriented applicationswith the capabilities of interacting via the publish/subscribeparadigm. In fact, the Event Handler core system is in charge ofthe notification of events that occur in a given Arrowheadcompliant installation, manages producers and consumers ofevents, allows filtering of messages, and manages historical dataregarding events. This latter capability is performed either onlocal files, on a database, or through another component of theArrowhead Framework - the Historian system. Two examples ofthe application of the Event Handler system are described: themanagement of application faults, and the support to quality ofservice of orchestrated services.

Extending publish/subscribe mechanisms to SOA applications

Michele Albano, Luis Lino Ferreira, José Sousa
CISTER, ISEP/INESC-TEC,
Polytechnic Institute of Porto
Porto, Portugal
{mialb,llf,jsous}@isep.ipp.pt

***Abstract*—The Arrowhead Framework is a European effort that aims to apply Service Oriented Architecture to the embedded systems' world. The Event Handler system is a component that supports the handling of events, and in that sense it enriches service-oriented applications with the capabilities of interacting via the publish/subscribe paradigm. In fact, the Event Handler system is in charge of the notification of events that occur in a given Arrowhead compliant installation, manages producers and consumers of events, allows filtering of messages, and manages historical data regarding events. This latter capability is performed either on local files, on a database, or through another component of the Arrowhead Framework - the Historian system. The net result of the integration of the Event Handler in an Arrowhead Framework simplifies and empowers the communication of its components, as it is demonstrated in the paper with two examples: the management of application faults, and the support to quality of service of orchestrated services.**

Keywords—components; system of systems; service oriented architecture; embedded devices

I. INTRODUCTION

The Arrowhead Framework [1] implements the grand vision of applying the Service Oriented Architecture (SOA) to the world of the Internet of Things [2]. To this aim, the distributed applications are supported by systems that compose systems of systems, and all interactions are based on the production and consumption of services. The Arrowhead Framework is currently supporting a large number of scenarios, spanning from virtual markets of energy [3] to management of intelligent elevators [4].

Even though SOA provides many benefits, such as the normalization of interaction processes [5], it imposes a rigid structure to the system of systems. The Event Handler system is a component that supports the handling of events, and in that sense it enriches service-oriented applications with the capabilities of interacting via the publish/subscribe paradigm. In fact, the Event Handler system is in charge of the notification of events that occur in a given Arrowhead-compliant installation, manages producers and consumers of events, allows filtering of messages, and manages historical data regarding events. This latter capability is performed either on local files, on a database, or through another component of the Arrowhead Framework - the Historian system. Two use cases involving the Event Handler system are described (Section V): the management of application faults, and the support to quality of service of orchestrated services [6].

978-1-5090-2339-4/16/$31.00 ©2016 IEEE

The paper starts with some background information on the Arrowhead framework, and on the publish/subscribe paradigm (Section II). Then Section III describes the applications scenario for the Event Handler, and in Section IV a detailed review of the services provided by the Event Handler is given. Section V wraps up the paper with a final discussion on the advantages of the Event Handler approach, and future work.

II. BACKGROUND INFORMATION

A. Arrowhead

The Arrowhead framework implements a Service Oriented Architecture approach by supporting local cloud automation functionalities [7] and offering a number of services that ease application development, among which discovery of services, loosely coupled data exchange between producer and consumer services, security-related services and orchestration of services.

The Arrowhead Framework offers the above mentioned functionalities through the definition of three groups of Core Services, Information Assurance services (IA), Information Infrastructure services (II) and System Management services (SM). It defines three mandatory systems, one belonging to each one of the three groups: Service Discovery (SD), Authorization and Authentication (AA) and Orchestration (O), to provide the services mentioned above. The SD system is used to allow service consumers to find the address of registered service producers. The AA system is used to authenticate and provide authorization for connections between services. The O system is used to determine the service producers that match specific criteria, e.g. choosing between services producers serving in the same geographical area where the service consumers are located. It is also responsible for the negotiation of QoS and keeping track of the system configuration. An example of its usage pertaining to home automation scenarios involves determining which services are capable of providing temperature readings in a house and to dynamically connect systems that need such kind of services with the most adequate providers of the service (e.g. according to the sensitivity of the readings).

Distributed applications involving systems and services can be visualized as systems of systems, which support the

distributed application. In the case of Arrowhead, the system of systems is structured in Local Clouds, which logically contain a set of application systems, and at least the set of mandatory core systems. Local Clouds can be supported by a single network infrastructure, where all nodes use the same technology, or by more complex infrastructure that includes multiple network technologies. The air-conditioning systems of a factory, composed by several temperature, humidity and dust sensors, air-conditioning devices and displays is a good example of a local cloud. The local cloud can be fully or only partially controlled by the Arrowhead framework.

Using this set of systems and their services, it is straightforward to design and implement a minimal local automation cloud. Fig. 1 shows an example which only shows the connection between application services (depicted in yellow). The application services are also consumers of the core services, depicted in red, green and blue.

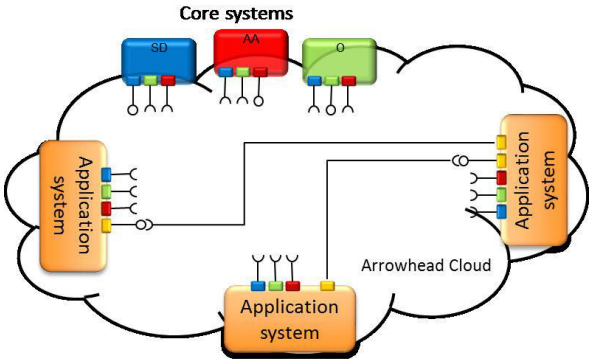

Figure 1 - Arrowhead application supported by the core services

The set Core Services offered by the Arrowhead framework is getting expanded, and new support systems are being developed, to be used together with the mandatory ones by the application systems. The set of support systems comprises the Historian and Configuration Manager, which carry obvious names. This paper focuses a new support system, the Event Handler system, which enriches the communication paradigms available to the application systems with Publish/Subscribe.

B. Publish/Subscribe communication

A limitation of SOA systems is that communications are based on a client/server paradigm, leading to a strong coupling of the involved parties. The Core Services of Arrowhead are usefully in facilitating the communication process, but there are scenario that need a stronger level of decoupling, in terms of space, time and synchronization [8]. Space decoupling means that publisher and subscriber do not need to be aware of each other's location or identities. Time decoupling means that publisher and subscriber do not need to be online and actively collaborating in the interaction at the same time. Synchronization decoupling allows asynchronous notification of subscribers by using event services callbacks.

A message broker, in our case the Event Handler system acts as an intermediary between the event producer and the event consumer, and leads to a paradigm providing asynchronous and highly scalable many-to-many communication model [9].

III. ARCHITECTURE OF THE SYSTEM OF SYSTEMS

The scenario enabled by this work is based on the introduction of the Event Handler system in the system of systems where distributed applications are being executed. The interacting systems are associated to the roles of event producers and consumers.

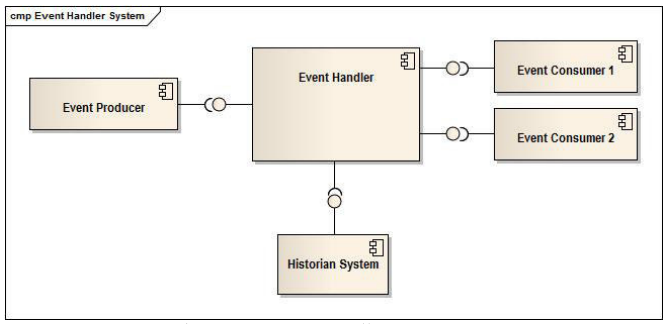

Figure 2 – Event Handler components

The Event Handler system provides functionalities for the notification of events that occur in a given Arrowhead compliant installation. It receives events from Event Producers and forwards them to subscribing Event Consumers (Fig. 2). The following list details the main actors for such a system and their roles: **Event Producer** is the component that creates an event and sends it to the Event Handler; **Event Handler** is the component that logs events to persistent storage, registers producers and consumers of event, applies filtering rules to event distribution; **Event Consumer** is the component that consumes the events, forwarded by the Event Handler; **Historian service** is an optional component used to store historical data on the events.

Depending on the context, an event can represent an exceptional occurrence on a particular system (e.g.: a value of a variable that reaches a critical level), or a simple change of state. Each event is classified according to a number of fields that represent the event's meta-data. An example of meta-data is the severity level of the event, which can hold the values: **Debugging** - information collected for debugging purposes; **Info** - tracing program execution: input/output data, changes in tagged variables, etc; **Notification** - state changes, execution of functions, etc; **Warning** - hint that "something might go wrong" in the near future; **Error** - malfunction in the system, application failure; **Critical** - severe malfunction of the system, which needs human intervention to continue its operation.

The Event Handler system has the intelligence of applying filtering rules to incoming events, based on the meta-data of the event (e.g.: severity level of the events), the system that produced the event, etc, to restrict the forwarding to the Event Consumers of events that are of their interest only.

Another function performed by the Event Handler is the storage of information regarding events, for future access. It occurs either on a filesystem local to the Event Handler system, or on a DB – which can be co-located with the Event Handler or on remote computer, or through a Historian service, which is another Arrowhead service used to store data [10].

Since the communication is service oriented, the Event Handler system receives feedback from the consumer

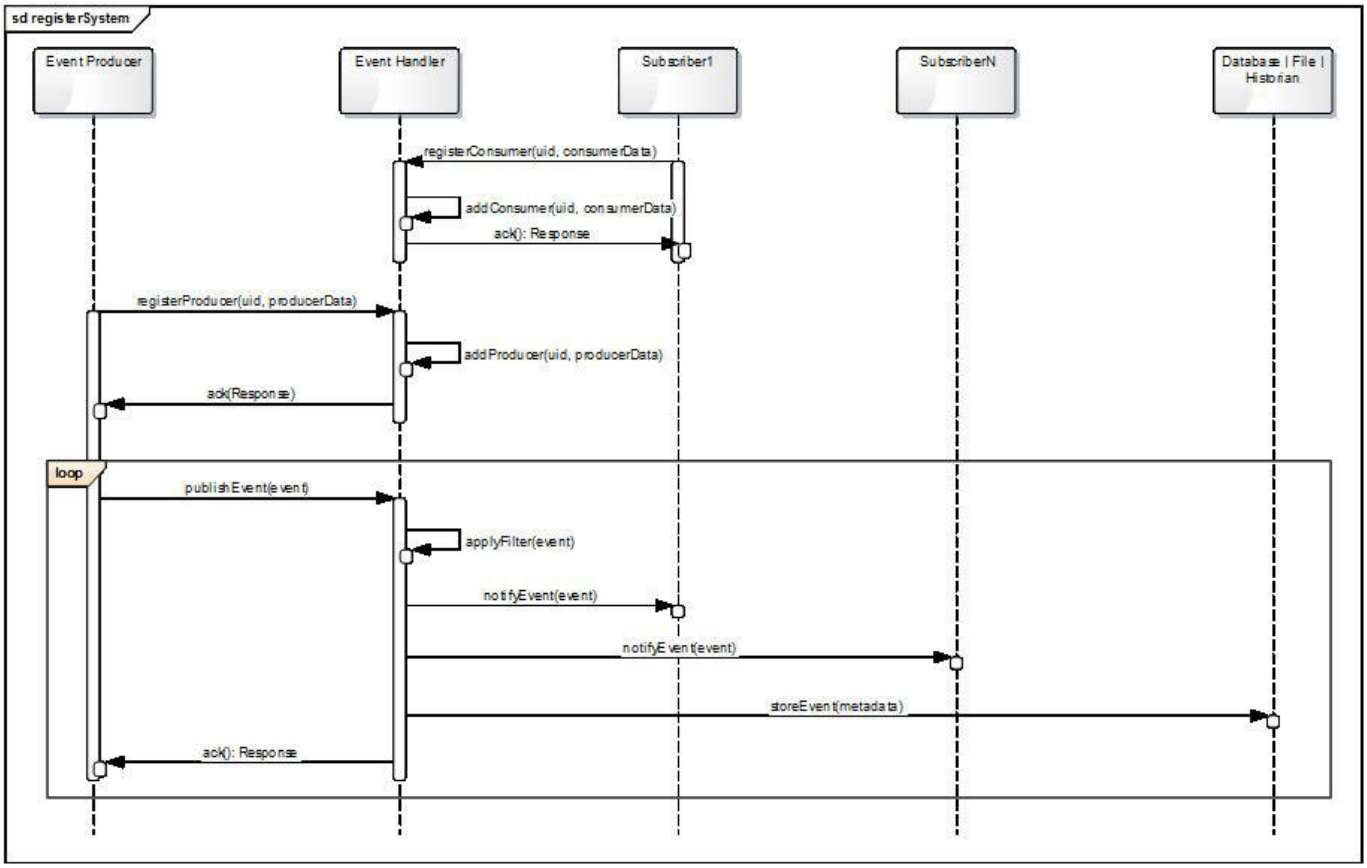

Figure 3 - Interactions between the systems

regarding the correct delivery of an event. When the Event Handler system does not receive positive feedback - or no feedback at all - it can consider that the event was lost. In this case, the Event Handler logs regarding the event indicate that the event was not delivered to the specific consumer; moreover, the Event Handler can generate an event - routed through itself - to parties interested in event delivery faults.

To implement these goals, the Event Handler workflow is enriched with a storage step, as described in Fig. 3. A number of Producers and Consumers register themselves, afterwards the producers send events to the Event Handler, which computes which consumers should receive the event, routes the event, and dumps the event on a storage area together with information regarding which subscribers received the event

IV. Structure of the Event Handler services

In the paradigm of Arrowhead, which is based on services, the structure of a distributed application is necessarily based on the description of the services it consumes and produces.

The Event Handler concept is built on 4 services – the Registry, Publish, Notify and GetHistoricalData services – that define all the operations that are performed in the context of the extension of the Publish/Subscribe paradigm to the SOA world.

The Event Handler Registry service is offered by the Event Handler system in order to store and keep track of all the consumers and producers in the system of systems. If a consumer wants to receive events, or a producer wants to publish events, they need to register through this service. In particular, at registration time the producer advertises the kind of events it produces; the consumer specifies the filtering rules regarding incoming events by defining a set of conditions that are applied to all incoming events, to compute which subscribers will receive each event. This service is also used to retrieve information on registered systems.

The Publish service and Notify service are used to deliver data regarding the events. A producer system accesses the Publish service of the Event Handler to provide the events it produces. The Event Handler computes which consumers must receive the event, and then accesses the Notify service of each selected consumer, to provide to it the incoming event. Direct consequence of this approach is that each event consumer must be a service provider, to allow the Event Handler to contact it when the latter has events to deliver.

The GetHistoricalData service applies filtering rules to permanently stored events (in a database, log file or through the Historian) and returns data regarding events. The service receives a Filter data type, which is analogous to the filter used when events are considered for delivery to event consumers. The returned data concerns the event under inspection, all the consumers that were contacted regarding the event on the first place, and whether each consumer received it or not.

V. Sample applications

Two different use cases of the Event Handler system are described here, demonstrating which compose its

functionalities with other Arrowhead services to close the circle on SOA embedded applications.

A. *Management of Faults*

The management of abnormal states for distributed applications can be tricky in the SOA world, since it involves providing a number of custom interfaces to allow other systems to collaborate in identifying application faults. The Event Handler is thus a storage point for logs regarding faults, and a routing element to allow monitoring system to receive online communications regarding faults from application services.

The example in Fig. 4 regards a temperature sensor whose data is being consumed concurrently by two applications. The sensor uses the Event handler to inform both applications that a fault has been detected, and to log fault details on a database.

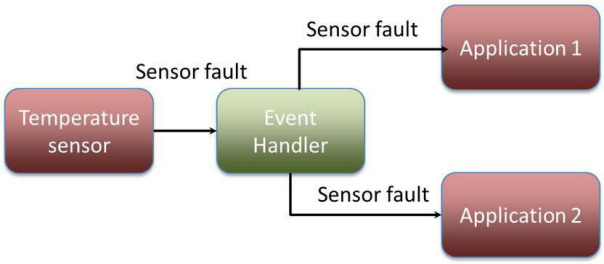

Figure 4 – QoS Monitor building blocks

B. *Support QoS through Orchestration Push*

In Arrowhead-enabled applications, service consumers can ask for QoS regarding service fruition through the Orchestrator system, which provides a matching engine between the requirements set by the service consumer and the capabilities stated by the service producer and the network connecting them. In this context, monitoring QoS can represent an excessive hurdle for service consumers, since these latter systems would need to provide means to receive asynchronous notification regarding QoS performance. An alternative is to connect a system performing the monitor of the QoS to the Orchestrator, via the Event Handler system.

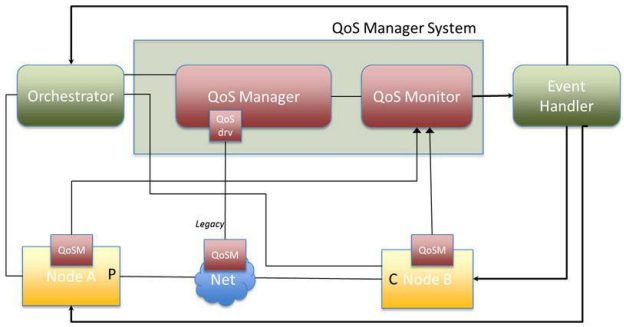

Figure 5 – QoS Monitor building blocks

Every time that the QoS is not respected, the QoS monitor will produce events, and the Orchestrator will receive them since it registers itself to the Event Handler every time it provides orchestrated services to a service consumer. Thus, the Orchestrator will be able to recognize the need to change the configuration of the System-of-systems on the fly. To this aim, the Orchestrator subscribes to QoS-faults messages, performs computation of a new configuration and pushes these changes to involved service consumers – every time QoS is not respected. Fig. 5 depicts the scenario discussed above.

VI. Conclusions

The Event Handler system proved to be a feasible solution to the problem of monitoring system events and conveying them to interested parties in a SOA scenario. In a more general sense, this paper provided an outline regarding the extension of publish/subscribe mechanisms to SOA applications.

Future work regards the measurement of the benefits provided by the Event Handler system, performed on real application scenarios. To this aim, both application services and core services in Arrowhead local clouds are getting integrated with the proposed system, to use its capabilities to manage events.

Acknowledgments

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0001/2012 - JU grant nr. 332987 (ARROWHEAD); and by the European Union under the H2020 Framework Programme (H2020-EE-2014-2015), EU ECSEL JU grant agreement nr. 662189 (MANTIS).

References

[1] Arrowhead Project, Enabling Collaborative Automation by Networked Embedded Devices, online at: http://www.arrowhead.eu/

[2] Derhamy, H., Eliasson, J., Delsing, J. and Priller, P., A survey of commercial frameworks for the Internet of Things. In Emerging Technologies & Factory Automation (ETFA), 2015 IEEE (pp. 1-8).

[3] Ferreira, L.L., Siksnys, L., Pedersen, P., Stluka, P., Chrysoulas, C., Le Guilly, T., Albano, M., Skou, A., Teixeira, C. and Pedersen, T., Arrowhead compliant virtual market of energy. In Emerging Technology and Factory Automation (ETFA), 2014 IEEE (pp. 1-8).

[4] Desdouits, C., Alamir, M., Boutin, V. and Le Pape, C., Multisource elevator energy optimization and control. In Control Conference (ECC), 2015 European (pp. 2315-2320).

[5] Rosen, Michael, Boris Lublinsky, Kevin T. Smith, and Marc J. Balcer. Applied SOA: service-oriented architecture and design strategies. John Wiley & Sons, 2012.

[6] Albano, M., Garibay-Martínez, R. and Lino Ferreira, L., Architecture to Support Quality of Service in Arrowhead Systems. INForum-Simpósio de Informática (INFORUM 2015).

[7] Colombo, A.W., Bangemann, T., Karnouskos, S., Delsing, J., Stluka, P., Harrison, R., Jammes, F. and Lastra, J.L., 2014. Industrial cloud-based cyber-physical systems. The IMC-AESOP Approach.

[8] M. Albano, L.L. Ferreira, L.M. Pinho, A.R. Alkhawaja, "Message-oriented middleware for smart grids", Computer Standards & Interfaces (2015), vol.38, pp. 133-143, Elsevier, DOI 10.1016/j.csi.2014.08.002

[9] P. T. Eugster, P. Felber, R. Guerraoui, A-M. Kermarrec, "The Many Faces of Publish/Subscribe", ACM Computing Surveys 35(2), 2003, pp. 114-131

[10] Pereira, Pablo Punal, Jens Eliasson, and Jerker Delsing. "An authentication and access control framework for CoAP-based Internet of Things." In Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE, pp. 5293-5299. IEEE, 2014.