Experiencing a SOA approach for network-centric data integration in the maritime surveillance domain

David Parlanti, Federica Paganelli
National Interuniversity Consortium for Telecommunications
Florence, Italy

Dino Giuli
Dept. of Electronics and Telecommunications
University of Florence
Florence, Italy

Agostino Longo
SELEX Sistemi Integrati
Rome, Italy

Abstract— Coordination and operational bodies involved in maritime surveillance need a common integrated maritime picture for performing their duties in an effective and cost-efficient way. Emerging trends for building an integrated maritime picture are focused on the adoption of a network-centric approach by leveraging on Service Oriented Architecture models and technological standards. This work presents results in the design and development of a message- and service-oriented middleware (named SAI – Service Application Integration – system), endowed with scalability, security and dependability capabilities. A demonstrator has been built in order to assess the SAI system capabilities for enabling information search and integration in the maritime surveillance scenario.

Keywords - maritime surveillance, service-oriented architecture, message-oriented middleware.

I. INTRODUCTION

The domain of maritime surveillance includes all the activities that could impact the security, safety, economy, or environment protection in the maritime sphere. It is well known that coordination and operational bodies involved in maritime surveillance need a common integrated operational picture for performing their duties in an effective and cost-efficient way. In the European Maritime Policy “Blue Paper” [1], the European Commission states the willingness to "take steps towards a more interoperable surveillance system to bring together existing monitoring and tracking systems used for maritime safety and security, protection of the marine environment, fisheries control, control of external borders and other law enforcement activities".

As highlighted in [2], existing systems are based on an “info-centric centralized approach”, where a Global Common Operational Picture is built on top of a central data repository system. While such an approach is well suited for addressing interoperability issues among relatively homogeneous communities, it is does not fit well with current requirements for the building of a cross-sectoral integrated information picture through the extension of the information network to a broader community.

In order to cope with these requirements, emerging efforts are focused on the adoption of a network-centric approach by leveraging on service oriented architecture models and technological standards.

This work reports on an ongoing study of the Dept. of Electronics and Telecommunications (University of Florence) and of the National Interuniversity Consortium for Telecommunications (CNIT) carried out in collaboration with SELEX Sistemi Integrati, aiming at experimenting and evaluating the adoption of well-fitting SOA models and technologies while coping with the maritime surveillance domain requirements in the european cooperative context.

We present results in the architecture, design and development of a message- and service-oriented middleware (named SAI – Service Application Integration – system), enabling information search, integration and delivery in the maritime surveillance scenario, which is a complex, distributed environment characterized by a significant technological and managerial heterogeneity.

This work is structured as follows: Section II provides a brief introduction to the maritime surveillance application domain and related interoperability requirements. While accounting such requirements. Section III analyzes the problem of systems and data integration in order to elicit a set of high-level architectural principles, according to a more general and application-independent perspective. In Section IV we describe the implemented SAI middleware overall architecture and briefly present its most relevant components. Section V discusses related work and Section VI reports on the demonstration activities carried out through a maritime surveillance case study. Section VII draws the conclusions and provides some outlines for future work.

II. INTEROPERABILITY REQUIREMENTS IN THE MARITIME SURVEILLANCE DOMAIN

The term maritime surveillance encompasses the wide variety of functional areas dealing with assuring maritime safety and security. More specifically, primary functional areas are the monitoring, control and enforcement actions for defense, search and rescue activities, maritime traffic control (including anti-terrorism and port security), environment protection, deterrence of illegal good traffic, drug traffic, illegal immigration.

In this domain, heterogeneous actors have different duties and responsibilities, depending on their institutional role. In the European Union area, which is our main reference for actual
demonstrations, most stakeholders may be classified into the following three layers:

- **Member State Coordination Layer**, grouping national level Ministries, such as the Ministry of Environment (MoE), Ministry of Interior (MoI), Ministry of Foreign Affairs (MoFA), Ministry of Transport (MoT).
- **Member State Operational Layer**, grouping Member state Operating Bodies such as Navy, Coastal guard, Port Authorities, Maritime Police, Antifraud corps.

According to current practices, these heterogeneous actors typically collect information for their own purposes by means of dedicated monitoring and surveillance systems. This situation may lead to many inefficiencies: similar information may be collected by different bodies, information which could be fully exploited by other actors (e.g. to detect cross-border threats) is not shareable (just to mention a few examples). In some cases actors may be unaware that potential useful information is collected by other actors, in other cases information sharing is not possible because of the lack of information sharing standards, agreements and policies.

As discussed in [2], the approaches currently adopted for building a common maritime situational awareness are commonly based on “info-centric centralized solutions”. In this kind of systems, an actor has the responsibility of collecting information in a central data repository system to build a global common operational picture and to broadcast it back to the affiliated organizations. While such an approach is well suited for addressing interoperability issues among relatively homogeneous communities, it is does not fit well with current requirements for the building of a cross-sectoral integrated information picture through the extension of the information network to a broader community. As a matter of fact, a large amount of information is considered as sensitive and owners want to retain control of the extent to which this information is shared with third parties.

Based on this background, our work aims at eliciting design guidelines and putting them into practice for the design and development of a service-oriented middleware and for coping with the requirements of complex, distributed environments characterized by a significant technological and managerial heterogeneity, as the one represented by the maritime surveillance domain.

More specifically, this domain is strongly characterized by the urgent need of a secured information exchange platform capable of bringing together existing and future heterogeneous monitoring and tracking systems for satisfying existing as well as evolving information needs of different actors in different functional areas (such as maritime safety and security, protection of the marine environment, fisheries control, control of external borders and other law enforcement activities).

In order to address the needs of such complex and variable interoperability scenarios our work aims primarily at targeting configurability and extensibility capabilities of the middleware. Secondarily, the work aims at addressing also requirements for assuring secured information exchange and basic system dependability.

### III. Systems and Data Integration for Network-Centric Operations

As already stressed by Thomas et al. [3], data integration is the typical problem for successful enabling of net-centric operations. System integration, in this area, is thus focused on the interfacing of legacy architectures, data formats and security solutions for allowing seamless data gathering and manipulation from multiple resources.

From a purely technical point of view, ad-hoc integration of multiple data sources can often be easily achievable through direct communication with legacy databases or data-sources, with little or no architectural concerns. This approach, however, presents significant drawbacks.

Firstly, the tight coupling between business logic and the internal structure of data silos makes the integrated system dependent on the mutable structure of managed data: as a consequence, any change in the data has ultimately to be reflected in the developed business logic, making the distributed system unmanageable in the long run. Secondly, such an approach usually requires the continuous polling of connected databases/data-sources for the retrieval of mission-relevant events. In this case, the risk is that of missing critical events during the polling interval, while also saturation of available network-bandwidth can occur whenever lot of data have to be transferred on the wire. Thirdly, the lack of standardized tools and languages for database management makes the developed system inevitably bound to a specific technology. Indeed, present tools for advanced data management always depend on specific technologies (e.g., the Hibernate ORM solution is only available on Sun’s Java or Microsoft’s .Net platforms), meaning that implementation choices will ultimately lead system design, as opposed to a structured top-down approach. Finally, managerial independence of defense departments can make direct access to data sources difficult, if not impossible, to negotiate. Such organizational constraints can be technically satisfied only by making explicit the trust boundaries of the connected organizations. As a consequence, an additional layer has to be introduced over direct database access for making explicit “on the outside” the organization security and data-disclosure policies.

By looking at such considerations, we end-up with a wish-list of architectural features that, in our opinion, all have to be satisfied for successful enabling of net-centric operations:

- the coupling of business logic with the internal structure of data should be avoided;
interaction of business logic with data should move from a pull (request/response) RPC-style of interaction to a push model;

- the interaction among system components should be based on open standards to avoid technological lock-ins;

- an indirection-layer over data should be introduced to structure interaction with data according to organizationally-defined policies.

The service-oriented architectural style (SOA) is a modern approach that can be exploited to meet the need for loosely coupled, event-driven, standardized and maintainable net-centric systems. The SOA is centered on the concepts of service, contracts, endpoints, messages, policies and consumers, as depicted in Fig. 1 [4].

![Figure 1. SOA main concepts [4]](image)

Services, the SOA basic system unit, can be concretely considered as processes that can be remotely accessed on specific network endpoints. Each service behaves accordingly with its associated interface, which has to be described by a publicly available “contract” through a standardized language. Contracts definitions are completely agnostic with respect to service implementation details. On the opposite, they are only meant to specify those operations and operands that can be successfully interpreted by any service implementation which is compliant with the contract specification. The functioning of a service is also governed by a publicly available policy specifying further interaction and non-functional requirements (e.g., the authentication scheme, QoS level and transport protocol supported by the service). On the consumer side, the SOA framework complements the basic service provision schema. Hence, access to service-managed data requires consumers first to understand syntax and semantics both of contracts and policies.

Being consumers capable to open network streams over the specified endpoint, interaction among service consumers and providers finally happens through messages. Messages are really the only mean of interaction among consumers and providers. Hence, as stressed by Helland [5], messages should also be durable to support failover mechanisms in case of system failures and restarts. This requirement has an evident architectural consequence, namely the need for a specialized subsystem, a “Message Broker”, tailored at managing application-level messages lifecycle and exchange patterns.

Helland [5] also suggests another important distinction for the SOA approach, that between Data Inside Service (DIS) and Data Outside the Service (DOS). DIS is data private to the service: it can only be accessed by service consumers according to the established contract and policy. In this regard, the service implementation acts as an indirection layer over data, and it shields organizations from malicious or unwanted use of internal information. DOS, on the other side, is transferred through messages among service consumers and service providers. DOS is usually correlated to DIS, but its format is established by the schema for operations and operands defined in the service contract. Since the service contract is expressed through a standardized language (e.g., XML), also DOS can be interpreted by consumers without any technological lock-ins.

IV. THE SAI ARCHITECTURE

The SAI middleware is conceived as a set of components which can be properly configured, assembled and extended in different system deployment configuration for enabling message exchange across environments characterized by managerial and technological heterogeneities.

The SAI architecture achieves such primary objective by grounding the service-oriented model on the message-oriented paradigm and by applying solid patterns in distributed systems design according to the SOA perspective described in the previous sections.

Instead of focusing strictly on performance and application-specific requirements, the work has been driven by the objective of primarily focusing achieving configurability and extensibility capabilities and supporting also basic security and dependability features.

Extensibility is intended as the capability of adding new features of modifying existing ones, while minimizing the impact on the existing system features and components. Configurability is intended as the capability of activating/deactivating and customizing system features in order to easily customize system deployments according to functional requirements and/or constraints of the technological environment.

SAI middleware design choices have thus been shaped to cope primarily with such non-functional requirements. To this purpose, we applied solid patterns in distributed system design and we adopted an programming with components approach.

More specifically, the SAI architecture applies SOA design principles to the problem of data-integration in the presented reference scenario, through the adoption of solid patterns [6], including:

- the “Message Broker” pattern as regards interaction among the system’s heterogeneous components;

- the “Adaptor” pattern for enabling uniform access to the orchestrated legacy systems;

- the “Master/Worker” pattern for enabling distribution and load-balancing of the system’s computational workload.
Current SAI implementation is largely based on the assembly of loosely-coupled and runtime-configurable components, which together provide a prototype for an integrated and dependable SOA system for data integration. Extensibility and configurability at functional and non-functional level is supported by the adoption of a component-based approach. To this purpose, we exploited the facilities provided by the Spring framework [7].

Spring is a layered Java/J2EE application framework, encouraging the adoption of good design and programming practices (such as programming to interfaces rather than classes). The Spring architecture is based on an Inversion of Control (IoC) container. Inversion of Control is a principle of framework design which aims at improving the reusability and extensibility of software by externalizing the creation and management of software component dependencies. As a consequence, the adoption of such framework facilitates the extension of a target system with new/modified features by adding/modifying “plain old java objects” (POJOs), i.e. software modules containing only business logic with no or little reference to the surrounding framework.

Security and dependability requirements have been addressed by developing and integrating dedicated components and by offering proper system deployment configurations (e.g. components clustering).

The logical structure of the SAI architecture (shown in Fig. 2) and the functionalities provided by each component are presented in the following subsections.

A. The Message Bus

The Message Bus is the infrastructure providing application-level messaging capabilities to the SAI system components. At present, the Message Bus component is powered by ActiveMQ, one of the leading open-source implementation of the JMS specification.

B. The Adaptors Framework

The SAI Adaptors Framework enables the interfacing of the SAI with heterogeneous information systems. Interfacing happens due to adaptation of the proprietary data format supported by legacy systems to the shared XML data model possibly used within a SAI-enabled application domain.

A SAI adaptor should be considered as a lightweight and configurable XML processor. Indeed, each Adaptor is a micro-container for a pluggable service implementation (see Fig. 3). An adaptor-managed service embeds the integration logic with an external information source (e.g. an embedded device or an enterprise-level system). In this regard, the service is a client of the legacy system, and it is charged of parsing received requests into the proprietary “dialect” spoken by the legacy system over the supported network communication protocol.

Each service can be completely described through its “functional profile”. The service functional profile describes the service’s processing capabilities by means of ordered input-output pairs of XML message types.

The adaptor micro-container manages the life-cycle of the service, while being capable to filter both incoming and outgoing XML documents through inbound and outbound interceptor chains. By exploiting such “interceptor pattern”, it is possible to augment the micro-container capabilities at runtime, simply by injecting additional interceptors in the inbound/outbound chains through configuration. As an example, interceptors have been implemented to enable compression of network streams, or to provide support for WS-* standards in order to free service implementations from unnecessary Web Services plumbing. The Adaptor is not bound to any specific XML envelope or format (e.g., SOAP), while support for XML standards can be configured at runtime through dedicated inbound and outbound interceptors. Security interceptors can also be developed to condition request processing or messages dispatching to authentication and authorization policies of legacy security systems.

Finally, an Adaptor can be configured to listen for request messages on a variety of network transport protocols and can support synchronous request/response, asynchronous one-way, asynchronous notification-response, solicit-response messaging patterns.

C. The Adaptors Registry

The “Adaptors Registry” component manages the “functional profile” of each legacy information system which is connected to the SAI system by means of a dedicated adaptor. A functional profile describes the message-processing capabilities of an adaptor through ordered input-output pairs of XML message-type identifiers and “meta properties”
(unordered name-value pairs) describing the non-functional capabilities of the mediated legacy system.

D. The Security Manager

The Security Manager is the SAI basic back-end service providing mechanisms for validating the identity of system’s principals, for granting or denying authorization for actions performed on SAI-managed resources, for guaranteeing the integrity and confidentiality of messages, and for supporting the evidence of actions performed by system’s principals.

E. The Transaction Manager

The Transaction Manager component is charged of coordinating global (distributed) transactions in the SAI distributed environment to ensure consistency of data access and manipulation operations. At present, the Transaction Manager component is powered by JOTM, which is an open and standalone implementation of Sun’s JTA specification.

F. The Grid Infrastructure

The SAI Grid Infrastructure provides workload distribution to applications and system services. SAI’s Grid has been developed from scratch in the Java Standard Edition for achieving full control and configurability of load splitting over system nodes.

Our implementation strictly follows the Master/Worker design pattern: at its heart, the grid simply consists of three entities: a master (that is the “client” of the grid), a channel for enabling master to worker communication and a set of one or more worker instances. According to such pattern, the master starts parallelization by defining a set of “jobs” which are then distributed (or “mapped”) to worker processes, then waiting for scheduled task to be completed. The final step then implies the master to organize (or to “reduce”) collected results into a “single” meaningful unit which shall be coherent with the semantics of the distributed work. While being adherent to the Master/Worker basic flow, the SAI grid also provides capabilities for: a) intelligent and configurable routing of jobs to workers; b) dynamic jobs reassignment in case of workers failures.

To provide such capabilities, the “PipeManager” component associates each worker instance to a unique “pipe” composed of a “pending jobs” and of a “completed jobs” queue. Being each worker linked to a unique pipe, jobs contention is minimized, while masters - which act as the clients of the grid infrastructure - can exploit pipes information to enable configurable jobs-routing algorithms. Being each pipe unique, and providing each pipe also the network endpoint of the corresponding worker, the PipeManager component can monitor the “liveliness” of each enabled worker and react to possible failures by re-assigning pending jobs on “live” executors. Such a dynamic reassignment of jobs is completely transparent to masters and live workers.

G. System Scalability and Dependability

System dependability can be defined as “the ability to avoid service failures that are more frequent and more severe than is acceptable” [8].

At present, the SAI framework achieves satisfactory dependability and scalability levels by means of: its Grid Infrastructure (which natively handles load-balancing and jobs-failover); the clustering of Basic Back-End Services and Adaptors; the implementation of basic autonomic capabilities to preserving the security state of the whole system.

As regards service redundancy, we distinguish between clustering of stateful and stateless services. Clustering of stateful services inherently requires state replication across activated service instances. In order to avoid failures due to inconsistent state, we have decided to first scale stateful services “vertically”, that is by running them on dedicated machines, while also starting background threads for replicating the “master” service instance state to secondary service instances in order to implement automatic switch-over policies in case of failure. Clustering of stateless services is easier because it does not require any state replication strategy. As a consequence, the level of redundancy that can be achieved on these components is arbitrary, while being limited only by the available hardware resources.

The SAI architecture also achieves basic support for autonomicity by exploiting the well-known “heartbeat” technique. Indeed, each SAI component can produce heartbeats that can be audited by other active components. Heartbeats help monitoring the overall system state. Moreover, when adapters stop hearing heartbeats from the security cluster they stop accepting requests and dispatching of outgoing messages until security heartbeats are resumed. This capability, although very simple to implement, still allow for fail-over strategies that are capable to preserve the security state of the system.

V. RELATED WORK

Among the wide variety of existing SOA-inspired middleware solutions, our work can be compared with some significant service-oriented systems based on a message-oriented paradigm in the maritime surveillance or military and defense domains.

In the european area, EuroSur [9] is an ongoing project aiming providing a common technical framework for promoting and improving the cooperation and communication among member states authorities for improving border surveillance activities.

SafeSeaNet [10] is a european project aiming at promoting information exchange across authorities and operational bodies involved in preventing and detecting illegal pollution actions. The SafeSeaNet solution is a centralized messaging system based on XML messages transported over the HTTP protocol.

The Net-Centric Adapter for Legacy Systems (NCALS) is a software prototype based on Java technology aiming at providing a cost-effective tool for integrating legacy systems on the defense domain according to a SOA-based approach [11]. Analogously to the SAI approach, NCALS applies the Adapter, the channel adapter and the message broker patterns, but to the state of our knowledge it does not provide support either for secured message exchange, or for workload distribution.
VI. DEMONSTRATION

In order to perform a qualitative assessment of the capabilities offered by the service-oriented SAI middleware, we conducted a demonstration activity in a test environment simulating the exchange of basic messages among legacy systems in the maritime surveillance domain.

The user can also access further information on displayed vessels, such as the current status (e.g. moored, at anchor, under navigation) the updated route plan, and pictures.

VII. CONCLUSIONS

This work has reported on the adopted approach and the experience in architecting, designing and developing a service-oriented middleware for enabling information sharing and surveillance services in the maritime surveillance domain, by a specific interpretation of guidelines for a SOA-based approach towards interoperability and integration in complex, heterogeneous and distributed environments. While often the SOA model is merely intended as a stack of Web Services standards, the SAI middleware design has been driven by SOA architectural principles in the form of design patterns [12].

The SAI middleware has been conceived in order to address first configurability and extensibility requirements, by adopting solid design patterns and component-based programming approach grounded on the adoption of the Spring framework. A demonstrator has also been developed and successfully trialed to assess most of the functional capabilities of the current SAI prototype in the maritime surveillance application domain. Nonetheless, we presently recognize the need for further specific benchmarking activities targeting the middleware performance profile. Benchmarking results will help us design additional infrastructure components and optimizations for achieving state-of-the-art levels of system resilience and scalability, while also suggesting minor refinements to the current prototype implementation.

REFERENCES