A Scalable Grid and Service-Oriented Middleware for Distributed Heterogeneous Data and System Integration in Context-Awareness-Oriented Domains

David Parlanti, Federica Paganelli, Dino Giuli, and Agostino Longo

1 Introduction

Context awareness deals with the capability of applications and services to react to specific events characterizing a target situation. The picture of such situation may be built by means of context information provided by sensors, i.e., physical and/or virtual sensors. Applications may exploit such situation awareness to react consequently, for instance, by enforcing adaptation actions at a logical layer (e.g., by sending notifications to interested users) and/or at a physical layer (e.g., by reconfiguring the physical environment by means of actuators).

This vision implies the integration of the physical world with the digital one and therefore requires proper instruments easing the integration of heterogeneous-embedded devices in ubiquitous computing environment (local scale) with enterprise-level services and business processes (global scale).

At present, we are investigating the adoption of an SOA (Service Oriented Architecture) approach for easing the integration of heterogeneous resources (e.g., sensors, actuators, enterprise information systems) for the development of context-aware applications in enterprise domains. We take as reference the domains of maritime surveillance and dangerous goods monitoring, where situation awareness pictures integrating local- and global-scale information resources are needed to develop added-value decision-support enterprise applications.

The SOA approach interprets distributed systems mainly as a problem of service specification, implementation, and composition. A "service" may be defined as a computational entity endowed with an open and addressable specification of

D. Parlanti (✉) and F. Paganelli
National Interuniversity Consortium for Telecommunications (CNIT), Italy
e-mail: david.parlanti@gmail.com; federica.paganelli@unifi.it

D. Giuli
Department of Electronics and Telecommunications, University of Florence, Italy
e-mail: dino.giuli@unifi.it

A. Longo
SELEX Sistemi Integrati SpA, Rome, Italy
e-mail: alongo@selex-si.com

D. Giusto et al. (eds.), *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications*, DOI 10.1007/978-1-4419-1674-7_11,

its expected behavior. We thus extend the definition of the "computation entity" to include software components encapsulating sensors/actuators functionalities. Integration of such real-world devices and business systems usually requires decoupling between service consumers and providers, thus demanding support also for one-way, notification response and solicit response interaction patterns. In order to address such invocation requirements, SOA's implementation solutions should be correlated also with "message-oriented" approaches. Message orientation gives new insights on service provision/consumption as well as on the overall SOA architectural style. More specifically, services can now be simply defined as "message-processors," while a service-oriented system can be consequently interpreted as a "network of connected message processors." Under this perspective, we present the SAI – Service Application Integration – system as a working example of a message-oriented SOA solution empowered with GRID scalability for data and system integration.

2 Related Work

This paragraph presents a few relevant works proposing middleware solutions for integrating sensor and actuator networks with web and enterprise applications.

Karnouskos et al. [4] propose a web-service device-to-business integration infrastructure by applying SOA principles to networked embedded devices. The solution is based on a web service approach, where each device offers its functionality as a set of services. Devices are attached to the middleware directly as web services using DPWS (via DPWS-enabled controllers) or by means of legacy system controllers.

The Global Sensor Network (GSN) middleware [1] is a solution aiming at providing a uniform interface for easing the integration and deployment of heterogeneous sensor networks. It is based on the abstraction of "virtual sensor," representing real sensors or software components aggregating different sensors in terms of input streams and one output stream. GSN adopts a container-based architecture. The container provides services for virtual sensors management, including remote access, security, persistence, and concurrency.

In [6], RESTful principles have been applied to elaborate a logical architecture of a middleware for enabling plug and play access to heterogeneous sensor and actuator networks, including addressing, discovery, and controlling mechanisms.

A proposal for a Service-Oriented Device Architecture (SODA) is presented in [3]. The objective of SODA is to integrate a wide range of physical devices into distributed enterprise systems, by providing the capabilities for accessing sensors and actuators as business services. A SODA implementation includes three main components: the device adapter, which translates proprietary and industry-based standard interfaces of devices into the device service abstract model; the bus adapter which maps the device service abstract model to the enterprise-level SOA binding mechanism; a device service registry providing discovery capabilities.

3 SAI Middleware Overview

The SAI middleware is targeted at enabling information search and data mash-up in complex, distributed enterprise environments characterized by strong technological heterogeneity. By "heterogeneity," we mean here differences in data-representation formats, data schema structure and semantics, communication protocols, security requirements and security-credentials management, and processing capabilities. The goal is that of making possible for clients to search data without any knowledge about its physical location in the distributed environment, while not worrying either about the consistency of data, even in case of failure of system components or of connected legacy information systems, nor about how the system internally distributes computing power or load-balances service requests.

3.1 SAI Architecture

The SAI architecture accounts for its primary system-level objectives through the adoption of solid patterns in distributed systems design, 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. A snapshot of the currently implemented SAI architecture is shown in Fig. 1. The logical structure of the architecture and the functionalities provided by each component will be presented in the following subsections.

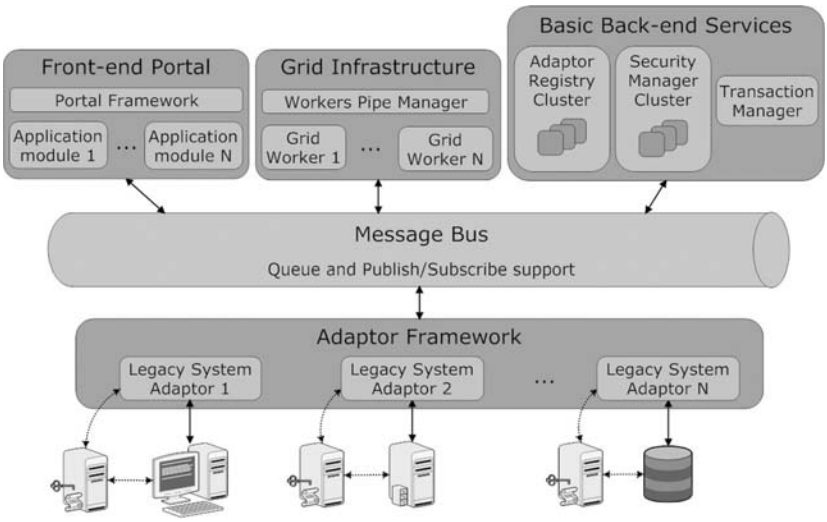

Fig. 1 SAI middleware architecture

3.1.1 Front-End Portal

The Front-End Portal infrastructure enables the uniform development of SAI front-end applications. The Portal Framework is a container establishing requirements for valid front-end "application modules," which can be composed at run time and delivered as a fully integrated web application and accessed via Desktop PC or mobile phone. At present, the Portal Framework component is powered by Liferay, an open-source implementation of the Portlet specification.

3.1.2 Grid Infrastructure

It is well-known that context aware applications usually need to process large amounts of data and that such processing needs parallelization to improve overall system performance, resilience and throughput. Accordingly, 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.

The SAI grid also provides capabilities for intelligent and configurable routing of jobs to workers and dynamic jobs reassignment in case of workers failures. To provide such capabilities, the "PipesManager" 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. Moreover, the PipesManager monitors the "liveliness" of each enabled worker and reacts to possible failures by reassigning pending jobs on "live" executors: of course, jobs dynamic reassignment is transparent both to masters and to live workers.

3.1.3 Basic Back-End Services

This group of components contains services providing cross-cutting capabilities supporting the functioning of the whole SAI infrastructure.

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.

The "Adaptors Registry" component manages the "functional profile" of each 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 nonfunctional capabilities of the mediated legacy system. The Adaptors Registry enables querying of registered functional profiles by authorized clients and monitors the operational status of activated adaptors to keep up-to-date the profiles registry status. It also listens for notifications concerning variations in the adaptors functional profiles.

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 the JTA (Java Transaction API) specification.

3.1.4 Message Bus

The Message Bus is the infrastructure providing application-level messaging capabilities to the SAI system components. The SAI interfaces are completely decoupled from any concrete messaging broker implementation, in order to enable maximum flexibility and scalability of the middleware. Indeed, messaging capabilities can be provided either by a full-fledged Message Oriented Middleware (MOM) or by lighter solutions based on IP multicast, such as the JGroups library. At present, the Message Bus component is powered by ActiveMQ, an open-source implementation of the Java Message Service (JMS) specification.

3.1.5 The Adaptors Framework

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

An SAI adaptor should be considered as a lightweight and configurable XML processor. Indeed, each Adaptor is a microcontainer for a pluggable service implementation. 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 microcontainer 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 microcontainer 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. Special security interceptors can also be developed to condition request processing or messages dispatching to authentication/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.

3.2 System Dependability

System dependability can be defined as "the ability to avoid service failures that are more frequent and more severe than is acceptable" [2]. At present, the SAI framework achieves satisfactory dependability levels by means of its Grid Infrastructure (which natively handles load-balancing and jobs-failover, as shown in the previous section), by the clustering of Basic Back-End Services and Adaptors and by the implementation of some basic autonomic capabilities for preserving the security state of the whole system. Indeed, the SAI architecture achieves basic support for autonomicity by exploiting the well-known "heartbeat" technique, since each SAI component can produce heartbeats that can be audited by other active components. Heartbeats help monitoring the overall system state and also enable automatic reaction of adaptors with regard to the state of the Security Manager service cluster. Hence, when adaptors stop hearing heartbeats from the security cluster, then they stop accepting requests and dispatching of outgoing messages until security heartbeats are resumed. This capability, although very simple to implement, allow for fail-over strategies that are capable to preserve the security state of the system.

3.3 Service Invocation and Composition in the SAI Framework

Each SAI information system adaptor is described by a functional profile that is stored in the Adaptors Registry cluster. Each adaptor is a message processor: its functional profile can be completely described by a set of unique XML input and output message-type pairs. Given such assumptions, Fig. 2a shows that message "M1" can be transformed into message "M2" through adaptor "A1" and that message "M3" can be transformed into message "M4" by means of adaptor "A2."

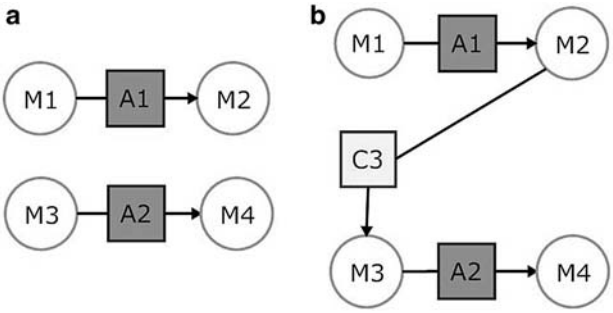

Fig. 2 Graph-based representation of service invocation and composition: (**a**) message "M1" can be transformed into message "M2" through adaptor "A1" and that message "M3" can be transformed into message "M4" by means of adaptor "A2"; (**b**) the connector "C3" links "M2" to "M3" through a syntactical transformation and it makes it possible to reach "M4" starting from "M1"

We can equivalently state that there is a "path" from M1 to M2 and another path from M3 to M4. When considered as a whole, the set of message nodes M1, M2, M3, and M4 and edges A1 and A2 together define a disconnected and directed graph "G," which is just an alternative representation for the information stored in the Adaptors Registry cluster state. It can be established whether a specific message can be transformed into another type of message by checking if a path between the two message types is found to exist in the graph-based representation provided by the Adaptors Registry. Such a path can be considered as the special "workflow" to execute in order to provide clients with the requested information.

Of course, situations may also occur where a same type of message can be produced by multiple adapters. For instance, we can suppose that M2 can be produced from M1 either by means of the A1, A2, or A3 adapter. SAI clients are then required to select among the available path from M1 to M2 by maximizing an objective function "f." The objective function is specified so as to take into consideration client preferences, as represented in their issued data-aggregation requests, and quality of service metrics as collected during repeated interaction with registered adapters.

In the SAI system, a "service invocation" simply consists of two elements: (1) a "request message," a message embedding information which is known or can be arbitrarily specified by a message producer; (2) the specification of a "target" message type to be produced by the SAI system, given the "request message" as input. Hence, the simplest request that clients can make to the system consists of a "target" message for which a direct path from the provided "request" message is found to exist in the adaptors registry. A complex request consists of a "target" message for which no direct path via a single adaptor invocation from the provided "request" message is found to exist in the adaptors registry. For instance, it turns out that a request for M4 given M1 is a complex request since there is no direct single connection between M1 and M4. A complex request could thus be handled by a path composed of multiple adaptors. In case that there are no registered adapters enabling M4 to be reachable from M1, from a logical point of view, a special entity ("connector") needs to be introduced in order to enable the system to respond to these types of requests.

A connector is an entity which is specialized in syntactical transformations (e.g., message-format adaptations) among message types. It is allowed to link message nodes with no outgoing links with message nodes with no incoming links. In Fig. 2b, we can see that connector "C3" transforms "G" into a connected graph, because it links "M2" to "M3" through a syntactical transformation: thanks to the connector, it is now possible to reach "M4" starting from "M1."

The injection of connectors into the SAI system, and the capability of clients to select one off multiple competing paths by means of special optimization algorithms, together allow the SAI system to solve complex data-aggregation scenarios. Moreover, since invocation workflows are created "on-the-fly" through simple operations on the graph-representation of adaptors profiles, there is no need for predefined hard-coding of invocation sequences into the system. Hence, as more connectors can be added progressively during the life-cycle of the system, SAI information retrieval capabilities can be said to be "evolutionary."

4 Case Study for Dangerous Goods Monitoring

We are currently performing some experimentation activities for evaluating the use of the SAI middleware for developing context-aware applications.

A preliminary case study for the SAI middleware has been designed in the framework of a research project in the domain of *european maritime surveillance*. The goal of the project is the provision of a secured information exchange platform capable of bringing together the 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 in order to satisfy information needs and information production profiles of heterogeneous actors (such as European Level Agencies Layer, Member State Ministries, Member State Operating Bodies).

Current research and prototyping activities are focused on a case study for *monitoring dangerous goods shipping across intermodal transport routes*.

Complexity of this application domain is due to several factors: the kind of transported goods and related risks for the surrounding physical and social environments; the heterogeneity of transportation means that are usually required for end-to-end delivery, the wide range of users which is involved to different extent in the shipping process. Most important user categories include [5]: transporters; final users (sender and consignee); multimodal operator; administration, authorities, traffic control services; emergency services. These users may be characterized by different regulations, specification, and technological infrastructure and pose different information needs requirements in terms of content type, information granularity, timing constraints.

We are currently designing and developing a service platform which, by leveraging on the capabilities provided by the SAI middleware, aims at providing information services to such different user categories. The middleware will support

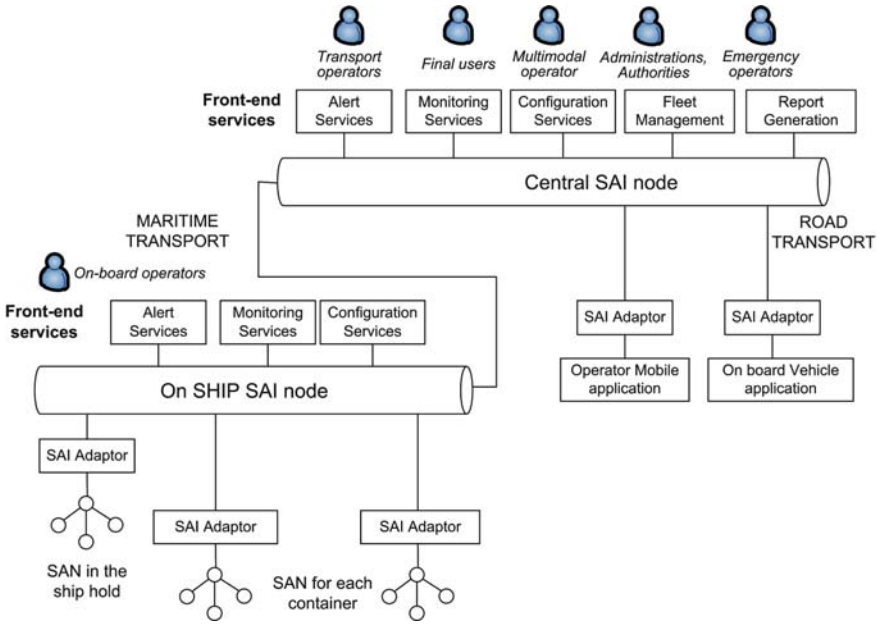

Fig. 3 Architecture of the dangerous goods monitoring case study

interoperability among the involved organizations and public authorities, while also hiding to clients the heterogeneities of the underlying technological infrastructures (e.g., sensors and actuators, on-board and mobile devices, enterprise-level and web-based application, etc.).

The prototype of the service platform has been customized to fit both the maritime transportation of dangerous goods and the interconnection with road transport segments. The service platform architecture is shown in Fig. 3.

The monitoring and control infrastructure is based on a set of RFIDs, sensors and actuator networks (SANs) which could be deployed in the container (or other unit loads) and/or in the ship hold in order to monitor and control the environmental conditions. The SANs expose their features as services via the SAI Adaptor (encapsulating SAN features as services to be exposed in the SAI architecture).

The back-end and front-end application logic which is deployed locally (e.g., on the ship on a local instance of the SAI node) and/or remotely (on a central SAI node) receives messages from the SANs. The on-ship monitoring service processes the information and if required, may trigger alert services to the on-board operators and/or to the remote monitoring system (e.g., via satellite communication). Analogously, SANs may receive reconfiguration commands for adapting their behavior according to a remote or local configuration adaptive logic (e.g., to increase the frequency of message delivery in case of abnormal conditions).

5 Conclusions

The SAI middleware shows that also enterprise-class technologies can be exploited in the development of context-aware applications. The proposed middleware has been empowered with a Grid infrastructure to offer workload distribution and load-balancing for the processing of context data, while still preserving the security state of the system and data consistency. Ongoing activities on the case study for dangerous goods monitoring in intermodal transport will help assessing the added value of the proposed integration system in a Web of Things scenario.

Acknowledgments This work was partially supported by SELEX Sistemi Integrati. Technical assistance from Luca Capannesi, Department of Electronics and Telecommunications of the University of Florence, is gratefully acknowledged.

References

1. Aberer K, Hauswirth M, Salehi A (2006) Middleware support for the "Internet of Things". In: 5th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze", Stuttgart, Germany
2. Avizienis A, Laprie J, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secure Comput 1(1):11–33
3. de Deugd S, Carroll R, Kelly K, Millett B, Ricker J (2006) SODA: Service Oriented Device Architecture. IEEE Pervasive Comput 5(3):94–96
4. Karnouskos S, Baecker O, de Souza LMS, Spiess P (2007) Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure. Proceedings of 2007 Emerging Technologies and Factory Automation Conference, pp 293–300
5. Nathanail T (1995) Architectural design for the monitoring of intermodal transportation of hazardous goods. Proceedings of the 1995 TransTech Conference, pp 69–73
6. Stirbu V (2008) Towards a restful plug and play experience in the web of things. Proceedings of the 2008 International Conference on Semantic Computing, pp 512–517