Enhancing message-oriented middleware capability and flexibility for dynamic service composition in context-aware distributed systems

Federica Paganelli, David Parlanti
National Interuniversity Consortium for Telecommunications (CNIT)
Florence, Italy
Federica.paganelli@unifi.it; david.parlanti@gmail.com

Dino Giuli
Dept. of Electronics and Telecommunications
University of Florence
Florence, Italy
dino.giuli@unifi.it

Abstract— Subject of this paper is the advancement of the middleware capabilities and configuration flexibility in complex context-aware distributed systems, according to the Service-Oriented Architecture (SOA) paradigm. Results are presented on an ongoing work concerned with the architecting, design and development of a message-oriented middleware for atomic and composite service brokering, named SAI middleware. The SAI middleware aims at flexibly offering a set of features for service brokering and dynamic composition across different application scenarios, while also guaranteeing loose coupling between service providers and consumers and relaxing the prerequisites for service providers to publish their capabilities in an interoperability domain. Our basic contribution consists in addressing these issues in a holistic way, as required to effectively support the SOA vision in context-aware application scenarios, while not optimizing single aspects yet, but demonstrating capability of easily matching requirements in different application scenarios.

Keywords: service-oriented architecture; service brokering; service composition; message-oriented middleware.

I. INTRODUCTION

Pervasive computing advancements aim at the realization of the concept of “smart” environments where computing and communication capabilities are distributed and seamlessly accessible to provide users with added-value context-aware services. Context awareness may be built on top of information provided by sensors, i.e. physical and/or virtual sensors. Applications may exploit system’s knowledge of relevant context information 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 re-configuring the physical environment by means of actuators).

This vision leads to 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 environments (local scale) with enterprise-level services and business processes (global scale) [25]. The potential advantages of sensor data availability for the development of novel business enterprise applications could be exploited in several application domains (tele-care, infomobility and goods monitoring, just to mention a few examples).

The problem of delivering data acquired from Wireless Sensor Networks to enterprise systems is a novel research area [30]. An emerging approach for easing the integration of heterogeneous resources (e.g., sensors, actuators, enterprise information systems, network capabilities) for the development of context-aware applications in enterprise domains is based on the adoption of the SOA (Service Oriented Architecture) paradigm.

The SOA approach interprets distributed systems mainly as a problem of service specification, implementation and composition [1]. A “service” may be defined as a computational entity endowed with an open and addressable specification of its expected behavior.

By adopting this view, the definition of the “computational entity” may be extended to include software components encapsulating sensors/actuators functionalities. Then the development of pervasive application may be designed as a workflow involving the invocation of services provided by heterogeneous components (i.e. enterprise-level systems as well as resource-constrained devices).

While the service-level abstraction may be profitably adopted for dynamically designing service composition flows integrating information coming from the above mentioned heterogeneous systems, the binding to concrete service providers and the concrete execution of such service flow must take into account heterogeneous constraints and capabilities of the system components (mobile devices, enterprise platforms, etc.).

As a contribution to the realization of this vision, our ongoing research activities are focused on the design and development of a novel service-oriented middleware, named SAI middleware. The SAI middleware aims at supporting the “publish, find and bind” SOA paradigm by providing artifacts for effectively decoupling service consumers from service providers and supporting dynamic invocation from client side of atomic and composite services, while relaxing service providers’ prerequisites for publishing their capabilities as services. While adherence to Web Services standards is a pre-requisite in several works [2],[5][12], our work aims at providing capabilities for relaxing such pre-requisite. As a matter of fact, Web Services specifications may require a large overhead and consequently cannot be always applied on resource-constrained devices in pervasive computing environments.

We model a “service” as an XML-based message processor. Systems which do not support Web Services
standards may be integrated into the systems via the intermediation capabilities provided by “Adaptors”. In the SAI middleware these components are lightweight and configurable XML processors which can be deployed in non-enterprise environments, such as the J2SE platform.

We are designing and experimenting the use of the SAI middleware along an evolutionary roadmap in research projects targeting different application domains, such as telecare home-based services for chronic patients, maritime surveillance and tourism context-aware guide. We are currently investigating advantages and issues in applying and extending the SAI middleware service brokering and composition capabilities for the integration of sensor and embedded devices for the development of a dangerous goods monitoring application.

The paper is structured as follows: Section II discusses related work. Service III provides a brief description of the overall SAI middleware architecture and Section IV describes the proposed approach for service brokering and composition. In Section V we conclude by discussing the presented results and providing insights into our recent or ongoing research activities aiming at adapting and experimenting the SAI middleware for the development of context-aware applications built on top of the integration of sensing and enterprise-level technologies in different application scenarios.

II. RELATED WORK

The issue of providing frameworks for dynamic brokering and composition of services has gained increasing attention in the last decade.

WSBinder [2] is a framework providing tools for dynamic binding of service compositions based on optimization of QoS criteria and functional policies. The framework takes as input a predefined composition process (e.g. a WS-BPEL workflow) and performs dynamic binding before and during the execution of the composition. Yu and Lin [3] propose a broker-based framework supporting dynamic service composition and QoS-based concrete service selection. Service brokering is supported by a semantic model and a semantic matching process in [4].

While addressing several relevant issues of Service Oriented Computing, the above mentioned works do not specifically address the need of effectively decoupling service consumers from service providers. A contribution in this direction has been provided by Daios (Dynamic and Asynchronous Invocation of Services Framework) [5]. Daios is a message-based service framework aiming at effectively allowing stubless service invocation and supporting asynchronous communication by adopting a message-oriented paradigm. The Daios invocation framework support both SOAP and REST-based services, however it does not aim at supporting the invocation of composite services.

To the state of our knowledge, state-of-the-art solutions aiming at addressing service brokering and composition issues while effectively decoupling service consumers from service provider, do not exist.

Several works focus on the proposal and optimization of service composition approaches and are typically based on workflow or AI planning techniques [6].

In workflow-based composition techniques the service composition problem is defined by means of workflow models. A possible approach may consist in defining at design time abstract process models and then dynamically select and bind concrete service implementations, such as in eFlow [7]. A dynamic approach includes typically both the creation of the process model and the selection of concrete service instances at runtime, as proposed in [8].

In AI planning techniques, service composition is interpreted as a planning problem. More specifically, the input and goal parameters in the request message are respectively modeled as the initial states and goals of the problem. Operations declared in known Web Service interface descriptions are modeled as actions permitted in the given domain. Several works have applied AI planning techniques to the composition problem. For instance, the work in [9] is based on situation calculus, while Mc Dermott proposes a composition solution based on the PDDL language [10]. Wu et al. in [11] present a method based on the use of the SHOP2 planner (a Hierarchical Task Network planner). Zheng and Yan [12] propose a syntactic-based composition algorithm based on planning graph model and a set of strategies for pruning redundant Web Service in order to optimize performance. Hewett et al. [13] propose an approach based on state space search model aiming at providing the shortest path solution and strategies for services pruning and simple parallelization.

Recently, a few relevant works have begun addressing specifically the issue of applying a service-oriented approach for integrating sensor and actuator networks with web and enterprise applications.

Karnouskos et al [25] 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 sys-tem controllers.

The Global Sensor Network (GSN) middleware [26] 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 [27] RESTful principles have been applied to devise a logical architecture of a middleware for enabling plug & play access to heterogeneous sensor & actuator networks, including addressing, discovery and controlling mechanisms.

A proposal for a Service-Oriented Device Architecture (SODA) is presented in [28]. 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 include 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.

Based on this state-of-the-art analysis, the contribution of the SAI middleware consists in providing a set of services for covering both service brokering and dynamic composition, while also guaranteeing loose coupling between service providers and consumers and relaxing integration prerequisites with respect to typical WS-based specifications. Typically such issues are addressed separately in order to focus on performance and optimization strategies. Our main contribution consists in addressing these issues in a holistic way, as required to effectively support the SOA vision in context-aware application scenarios, while not optimizing single aspects yet.

III. SAI MIDDLEWARE

This work proposes a middleware providing mechanisms for easing the design and development of client applications accessing services provided by heterogeneous providers. More specifically, main requirements consist in:
- effectively decoupling service consumers and service providers and providing consumers with a uniform API masking differences among service providers in formats and protocols for service discovery and invocation;
- relaxing the prerequisites for service providers to publish their capabilities and make them invokable from the outside (i.e. compliance with WS specifications and, more specifically, to a predefined style of WSDL should not be a prerequisite for integration);
- supporting dynamic service composition.

The SAI middleware is a service- and message-oriented middleware that can be properly configured, assembled and extended for enabling XML message exchange across environments characterized by managerial and technological heterogeneity. It is has been conceived and iteratively designed and developed in the context of previous research projects for coping with interoperability requirements in different applications domains, such as maritime surveillance, home-based care assistance and dangerous goods monitoring. For the sake of completeness, we provide here a brief description of the SAI middleware capabilities. More details on the middleware and its experimentation on the above mentioned domains are reported in [15], [14], [16].

A. SAI design principles

The SAI architecture achieves its primary objective by grounding the service-oriented model on the message-oriented paradigm and by applying solid patterns in distributed systems design [24] according to the service-oriented perspective.

Instead of focusing strictly on performance and application-specific requirements, the SAI middleware design has been driven by the objective of achieving configurability and extensibility capabilities.

In this direction, most relevant patterns adopted in the SAI middleware design include:

- the “Message Broker” pattern, which has been applied for handling the interaction among the system’s heterogeneous components as well as among cooperating external systems;
- the “Adaptor” pattern, adopted for enabling uniform access to the services offered by 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 is also supported by the adoption of abstraction by interface and component-based programming approaches. To this purpose, we exploited the facilities provided by the well-known Spring component framework [17].

B. Enabling Interoperability through Message-Based Service Invocation Handling

The SAI middleware aims at facilitating interoperability and information exchange across heterogeneous systems in target application domains by leveraging on the message-oriented paradigm. More specifically, the SAI middleware is a configurable and extensible message-exchange platform intermediating service consumers and providers interactions (such as service selection, binding and invocation) while also minimizing their reciprocal dependency.

Most common WS-support client libraries typically rely on the generation of static stubs generated at design time from WSDL descriptions, thus binding the client to the provider specific interface. On the contrary, the SAI middleware enables a client application (referred to as service consumer) to query and access information made available from different providers via XML message passing.

Instead of being programmatically bounded to operations and input parameters exposed by service providers, clients may issue requests by sending a message to the SAI middleware. The client message is decoupled from the service provider interface, as the SAI middleware is in charge of interpreting the request and selecting the proper atomic or composite service to be invoked. The result produced by the service provider is properly elaborated in a response message to be sent back to the client.

The message exchange with the client interface is handled by a messaging interface, named “DeliveryChannel”. The DeliveryChannel may offer different interfaces for message passing, such as XML messages over HTTP and JSON over HTTP.

The minimum constraint for a valid XML message request is to comply with the following structure:
At present, interoperability in a specific application domain is handled at syntactic level by allowing to inject an application-dependent data and messaging model defined in XML Schema in the system.

More details on the service brokering and composition approach in the SAI middleware are provided in Section 5.

On service providers side, the SAI system offers two ways for publishing and making services accessible:

- service providers may expose their capabilities as Web Services described in WSDL documents. In the tourism demonstration scenario, we assume such WS interfaces to be compliant with XML elements specifications encoded in the SAI application messaging schema mentioned above. Other cases could be handled by manually building transformation rules (e.g. XSLT files), while for minimizing human intervention we should enrich the middleware with semantic and matchmaking capabilities, as discussed in the Conclusion section.
- Service providers which do not support WS specifications, may deploy a component provided by the middleware, named SAI Adaptor component, which can host the ad-hoc logic for translating its custom application interface into the SAI application messaging schema. The SAI Adaptor can be deployed leveraging on a Java 2 Standard Edition Platform and thus not requiring heavyweight enterprise environments.

C. Main SAI Components

As depicted in Fig. 1, the middleware architecture is based on the following core components:

- A Message Bus, offering messaging exchange capabilities among SAI internal and external components, and a general-purpose message-handling interface, named Delivery Channel. The Delivery Channel can be specialized for allowing clients to access application-dependent information exchange capabilities in a given interoperability application domain. The SAI Message Bus is completely decoupled from any concrete messaging broker implementation and a messaging system adaptation layer can be properly configured to host adaptation modules towards different messaging brokers implementation and deployment configurations. At present, the Message Bus component is powered by ActivaMQ, one of the leading open-source implementation of the JMS specification (Java Message Service [20]).
- Adaptors, which are containers hosting the integration logic for properly intermediating the access to heterogeneous legacy information systems, according to the vocabularies and rules defined in a given interoperability application domain. More specifically, the integration logic is in charge of translating messages formatted according to a shared XML data and message model into the legacy systems' specific data formats and protocols. An adaptor can be customized with proper message processing modules enforcing legacy systems specific policies (such as security and data sharing policies) along the message flow.
- A service registry, which maintains the interface description of registered services for information service discovery, invocation and monitoring.
- A Transaction Manager component, for ensuring consistency of SAI-managed data access and manipulation operations.
- A Security Manager, which is the back-end component providing authentication, authorization, confidentiality, integrity and non-repudiation services.
- A Composition Manager, which handles requests for composite services by applying dynamic composition techniques (as described hereafter).
- A Grid Infrastructure, providing workload distribution capabilities for resource intensive processing tasks.

IV. SERVICE BROKERING AND COMPOSITION

Service consumers can leverage SAI messaging capabilities by exposing a WS-based interface or deploying a proper adaptor. In both cases, service interfaces have to be published in the SAI system registry. According to our approach, a service is essentially seen as a message processor: hence, its interface is described by a functional profile defined as a set of XML input and output message-type pairs and a set of metadata representing non-functional properties (see Section 5.1).

When a client invokes a service, the “request” message in the SAI middleware should contain the following two elements:

- Expected Data, i.e. the “target” message type expected as response from the SAI system and/or expected output parameters names;
- Provided Data, i.e. input parameters for specifying the request and producing a valid output;

The request is received by the SAI system and analyzed in order to find proper solutions, i.e. the adaptors that could
provide the target response. As a consequence, depending on the functional profiles registered in the SAI registry, a request may: i) be handled by invoking a basic service (i.e. a single operation offered by a service provider); ii) when a matching basic service does not exist, be handled by invoking a composite service, i.e. a service specified by composing other basic or composite services; iii) have no known solutions in the SAI domain.

When a request message is received, the SAI system performs the following activities:
- querying the service registry for finding a matching basic service or a plan defining a composite service;
- binding the solution plan to concrete adaptor endpoints (or a single adaptor in case of a basic service solution);
- executing the solution plan by invoking the selected services (via SAI adaptors and/or WS-based service invocation).

At present we have implemented a service composition engine based on AI planning techniques (see Section 5.2). Configurability capabilities of the SAI middleware may be applied to support alternative dynamic composition techniques.

A. Ontology-based Service Profile model

Functional profiles are represented in the Registry via an ontology-based model written in the Web Ontology Language (OWL) [19]. Fig 2 shows main entities and properties encoded in the OWL-based functional profile model by means of UML class diagrams. UML Classes represent OWL classes, attributes represent OWL Datatype properties and associations among classes are used for representing OWL Object Properties.

The entities defined for representing services in the Registry domain are distinguished into two disjoint classes: StructuralEntity and QualifyingAttribute. StructuralEntities include the basic constructs for describing a functional profile: Atom, Message, and Operation.

An Operation defines the functional capability of a service as a black box that requires a Message as input and produces a Message as output. A Message is characterized by a qualified name (has_qname). By assuming a tree-based structure of a message (assured by any well-formed XML document), the Atoms represent the message attributes and leaf elements typically carrying literal values. Atoms are identified by a QName (has_qname). The attributes min_occurrence and max_occurrence specify if the Atom is required and its allowed multiplicity.

The attributes has_extraction_path and has_structural_path carry information extracted from XML Schema message definition and which can be useful, respectively, for dynamically extracting message parts via XPath queries and constructing message parts taking into account XSD-specified constraints (such as type of content model and element order in a sequence).

QualifyingAttributes are entities used for further characterizing StructuralEntities, i.e. by adding non-functional properties such as reputation, cost, and QoS properties. At present, we have defined a general-purpose Property simply characterized by a (name, value) tuple.

This models enables to represent messages expressed as XML document templates or XML-Schema based specifications. Our current model does not fully cover the whole XML Schema specification, but the basic constructs, such as “sequence”, “all” and “choice” content models, elements multiplicity, attributes and their optional/required use.

![Figure 2. Example of structural path and extraction path expressions obtained by XML Schema parsing](image)

The SAI middleware implements parsing features taking as input adaptor profiles and, similarly, WSDL-based interface descriptions and automatically generating the service functional profile according to this ontology-based model. These parsing features has been implemented by extending the XSOM (XML Schema Object Model) library [18].

1) Atom data manipulation

Mechanisms for effective XML data manipulation are needed for implementing service brokering and composition services. As described in the following section, a composition plan execution requires proper actions for dynamically extracting data from output messages and combining them into input messages.

The attribute has_extraction_path is a string-based representation of the position of an Atom in the embedding Message, according to the XPath syntax.

The attribute has_structural_path is similar to the extraction path. In addition, it represents information needed for dynamically reconstructing part of messages valid according to the XSD-encoded definitions. More specifically, we have defined a custom numerical code specifying whether an element is contained in a “sequence”, “all”, “choice” compositor and, consequently the order in a sequence and disjointness relationship in a choice.

Fig. 3 provides an example of the structural and extraction path expressions that can be associated to the
“reservedDate” atom data in a message, named HotelReservationResponse, defined with XML Schema constructs.

Figure 3. Ontology Model for the functional profile

B. Service Composition

The problem of service selection and composition can be specified as a query containing known input parameters values and target output parameters names.

In the SAI system a functional profile is defined in terms of input/output XML message types. Message types definitions (for instance XML Schemas) specify how message types are structured and which inner elements (e.g. input/output parameters) are contained. In the following we will thus interchangeably refer to parameters and/or message types, as message type names can be straightforwardly associated to the set of contained atom data (i.e. input/output parameters).

By matching profiles’ input and output parameters it is possible to build a graph of connected functional profiles. A functional profile may be added to a chain of connected functional profiles, if all required input parameters of that profile are within the output parameters of profiles already present in the chain. The new profile thus may be connected to such previous profiles. The problem of service solution may thus be stated as finding a chain of functional profiles whose input parameters are within the query known parameters and whose output (message) parameters contain the query target (message) parameter.

By stating the problem in this way, it is quite straightforward to apply AI planning techniques to find service composition solution plans [6]. More specifically, we adopted the Graphplan algorithm [21].

The Graphplan planner has been first proposed in [22] to provide an effective way of building plans in STRIPS-like domains. These domains consist of initial conditions that describe the starting state of the world, operators which describe the actions that may be performed, and goals representing the state to be reached. Operators have preconditions, add effects and delete effects. The Graphplan algorithm compile the problem into a structure called planning graph. A planning graph is a directed, leveled graph where the first level contains the initial conditions, the next levels have a node for each action that might possibly be performed (that is whose pre-conditions all exist in the previous level) and so on. The Graphplan begins by creating a planning graph in a forward direction from the initial conditions and then expands itself one level at a time until a solution is find. More specifically, at each time step, the algorithm searches in a backward-chaining fashion if the goals appear in the graph. If the recursive search does not find a plan, then the graph is extended one more time step.

We thus have formulated the service composition problem as a planning graph. Initial conditions are represented by the query input parameters, the goals by target output parameters and operators are represented by functional profile operations. For each operation, input message are modeled as preconditions and output messages as add effects. Moreover, as messages contain inner parameters, as defined in corresponding XML Schemas, the STRIPS domain is also populated by further operation types representing syntactic containment relationships.

![Figure 4. Representation of a service composition problem in the STRIPS domain based on the Graphplan algorithm](image)

Fig. 4 provides an example of the composition approach adopted in the SAI middleware. Table 1 contains a list of functional profiles and names of related atoms in input and output messages (a letter is assigned to each profile and atom name in order to make the graphical representation in Fig. 4 more compact). Fig. 4 provides a simplified example of a Graphplan-based approach for solving a composition problem where a, c, f and g parameters are given as initial conditions and h is the target parameter (goal).

<table>
<thead>
<tr>
<th>Functional Profile</th>
<th>Atoms in Input Message</th>
<th>Atoms in Output Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUserLocation(p1)</td>
<td>userId (a) userLocation (b)</td>
<td></td>
</tr>
<tr>
<td>findOptimalHotel(p2)</td>
<td>userPreferences (c), userLocation (b) hotelId (d)</td>
<td></td>
</tr>
<tr>
<td>reserveHotel(p3)</td>
<td>hotelId (d), date (e), nightNumber (f) roomType (g) reservationResponse (h), reservedDate (i), reservedNightNumber (l), reservedRoomType (m)</td>
<td></td>
</tr>
</tbody>
</table>

At present we have built a STRIPS domain builder which parses interface descriptions maintained in the SAI service registry for creating corresponding actions. The Graphplan
algorithm implementation is based on the open source PL-PLAN library [23].

V. CONCLUSIONS

This paper has reported on current results of our research activity in the design and prototypical development of the SAI middleware. SAI is a novel service- and message-oriented middleware that can be properly configured, assembled and extended for enabling XML message exchange across pervasive environments characterized by managerial and technological heterogeneities.

The SAI middleware provides a set of services for covering both service brokering and dynamic composition, while also guaranteeing loose coupling between service providers and consumers. Our main contribution consists in addressing these issues in a holistic way, as required to effectively support the SOA vision in context-aware application scenarios, while not yet optimizing single aspects. More specifically, SAI advantages illustrated in this paper consist in:

- Loose coupling between service consumers (i.e. the client application) and service providers. As a matter of fact, stub generation based on service providers’ WSDL description is not required, as clients may invoke services via the DeliveryChannel message-handling interface. The DeliveryChannel may be customized depending on the target application domain in order to provide clients with a uniform API based on an application-dependent XML data and message model.
- Relaxed prerequisites for service providers to publish their capabilities and make them invokable from the outside. Provider capabilities can be exposed by publishing a standard WS-based interface or by customizing a lightweight adaptor component.
- Dynamic service composition; at present the SAI approach for dynamic composition is based on AI planning techniques and on an ontology-based functional profile model encoding information needed for automatic XML message parts extraction and combination.

Further activities are planned to enhance SAI service brokering and composition capabilities. At present service brokering and composition are based on syntactic matchmaking. Compliance with an application-dependent data and message XML-based model is required for effectively employing SAI service brokering and composition capabilities and for guaranteeing interoperability among service consumers and providers. In order to widen the interoperability scope, we are also investigating the use of semantic annotations in service descriptions (e.g. SAWSDL) for enabling semantic matchmaking via a shared application-dependent ontology. Finally, no specific service provider selection strategies have been developed yet. To this purpose, we are planning the use of QoS-aware selection strategies for global and local optimization.

SAI middleware capabilities for facilitating the development of distributed applications in complex environments are being experimented in different application scenarios in the context of research projects participated by the CNIT Research Unit at the University of Florence.

Firstly, in the framework of a study performed by the Dept. of Electronics and Telecommunications (University of Florence) with the National Interuniversity Consortium for Telecommunications (CNIT) and carried out in collaboration with SELEX Sistemi Integrati, we experimented and evaluated the adoption of SOA models and technologies to cope with interoperability requirements in the maritime surveillance domain. In this context, we experimented the adoption of the SAI middleware as a secured information exchange platform integrating information acquired by heterogeneous monitoring and tracking systems for satisfying existing as well as evolving information needs of 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) [15].

Secondly, we are involved in the ASINFO research project, where the SAI middleware is evolving to provide an extensible service-oriented platform for the development and delivery of location-based and context-aware mobile services to tourists [29].

Thirdly, as follow-on activities of the KAMER research Project [14] we have applied the SAI middleware in a home-based telecare scenario. We have thus experimented the use of the SAI middleware as a service-oriented framework for continuous care, providing services for information acquisition, integration processing and sharing among care operators, for patient and home monitoring, alarm detection and alert management.

Finally, ongoing research activities are investigating the adoption of still enhanced service brokering and composition capabilities of the SAI middleware on a case study for dangerous goods monitoring across intermodal transport routes, within the SITMAR research project framed within the “Industria 2015” national research program.

The latter application scenario is characterized by a monitoring and control infrastructure made of RFIDs, sensors and actuator networks (SANs) to be deployed in the container (or other unit loads) and/or in the ship hold for assuring environmental and goods conditions monitoring and control. An on-ship monitoring service hosted by the SAI middleware processes sensor data and eventually triggers alert services to the on-board operators and/or to a remote monitoring system (e.g. via satellite communication). This case study represents a significant scenario for eliciting requirements and experimenting a service-oriented approach for the development of pervasive applications based on the integration of sensor and enterprise-level technologies. A special effort will thus be devoted to the development of adaptor integration logic for handling event streams produced by sensor networks.

REFERENCES


[18] XML Schema Object Model (X SOM) https://xsom.dev.java.net/


[22] A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis", Artificial Intelligence, 90 (1997), 281—300


[24] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1995


