On the modeling of dynamic reconfiguration of embedded service oriented applications

A. Armentia*, E. Estévez*, I. Sarachaga*, M. Marcos*

*Departamento de Ingeniería de Sistemas y Automática, ETSI de Bilbao, Universidad del País Vasco (UPV/EHU), España
(e-mail: {aintzane.armentia, elisabet.estevez, isabel.sarachaga, marga.marcos }@ ehu.es)

Abstract

Service Oriented architectures are becoming very popular to meet the requirements of complex current applications demanding distribution and heterogeneity in open environments. The loose coupling provides reusability and interoperability, avoiding platform dependencies and saving time and development costs. The iLAND-ARTEMIS project goes a step further aiming at the development of a service-based middleware framework capable of supporting deterministic dynamic functional composition and reconfiguration of networked embedded applications. In this context, this paper proposes a Model Driven Approach to support the composition and dynamic reconfiguration of networked embedded service oriented applications.

1. Introduction

Over the last years software architectures are getting more and more complex demanding the development of more adaptable applications. As traditional architectures do not seem to be capable of dealing with these new requirements (distributed embedded systems in open and heterogeneous environments), the service-oriented architecture approach is becoming widely adopted, mainly in applications running on varied technologies and platforms that have to communicate with each other.

The service-based paradigm allows developing software applications as collections of loosely coupled services that interact [1]. It is more flexible than other traditional approaches for software design, providing reusability and interoperability, avoiding platform dependencies and saving cost and time during the development of the application [2].

Although it is an emerging paradigm, composition of services has received much interest to support business-to-business (B2B) or enterprise applications integration. Two are the main approaches that have been developed: Web and Semantic Web Services. In [3] an overview of existing service composition languages for both approaches is compared against the requirements that a service composition language should support to facilitate business processes composition.

The work described in this paper is based on the use of the service-based paradigm in networked embedded systems aiming at supporting dynamic reconfiguration. This approach has been proposed in the context of the ILAND (mIddLewAre for deterministic dynamically reconfigurable NetworkeD embedded systems) project [4]. The main objective of the project consists of developing a component-based middleware framework capable of supporting deterministic dynamic functional composition and reconfiguration of distributed applications [5][6]. As a result, an improvement on system flexibility, scalability, and composability are achieved. Also, spontaneous reconfiguration of the system is also supported; hence, the maintainability is improved. It enables dynamic functionality reconfiguration, e.g. node activation, removal of crashed or damaged nodes and re-allocation of functionality.

The iLAND middleware components need to manage information in order to provide support to composition and reconfiguration of SOAs (Service Oriented Applications). Therefore, modeling languages that allow system description and definition before construction must be used. They also guide the definition, composition and reconfiguration of a set of SOAs. The Model Driven Architecture (MDA) standard proposed by OMG [7] separates the system specification (Platform Independent Model, PIM) from the implementation issues (Platform Specific Model, PSM). MDA is being widely applied to SOA architectures [8], [9] and [10]. The Model Driven Development (MDD) goes a step further. It relies on the use of models to represent the system elements of the domain and their relationships [11].

The Model Driven Approach has been applied for designing and implementing (code generation) of
distributed systems as in [12] and the MODELPLEX project [13]. In the area of embedded systems, some model driven work has been done, mainly for special domains. For example, the work done in [14], the DECOS project in the automotive and avionics field [15], and the MEDEIA project in the industrial automation sector [16].

On the other hand, domain specific modeling languages have also been developed for distributed and/or embedded systems such as [17]. The society of automotive engineers (SAE) developed the SAE Architecture Analysis & Design Language (AADL), initially for aerospace applications [18]. In [19] a tool suite built around the AADL is proposed in order to support the different steps of system construction in DRE projects, from early prototypes to final implementation. For example, the TIMMO project aimed to develop a domain specific modeling language for handling timing information while developing automotive distributed embedded systems, the TADL [20].

The application of these concepts to iLAND applications leads to the definition of models containing the relevant data that the iLAND middleware components need to manage.

The layout of the paper is as follows: Section 2 describes the proposed modeling approach for the iLAND system. Section 3 presents examples of iLAND applications as a proof of concept of the modeling approach. In Section 4 a UML-based modeling language is presented. Finally, Section 5 presents the future work and conclusions.

2. Modeling Approach

The modeling mechanisms must offer enough expressive power in order to provide means for specifying the functionality of iLAND applications, including application logic, application needs of reconfiguring its functionality at run-time, and the QoS the application demands to the middleware. Furthermore, the application model must capture all the information needed by the components of the iLAND middleware to create and reconfigure applications at run-time assuring that both, functional and non functional requirements of all existing running applications are met. Thus, it is also necessary to characterize system resources from both, platform independent and platform dependent point of view.

iLAND models should be an abstraction of the system characteristics that are relevant for the different stakeholders that interact during system operation: the user and the iLAND middleware components. In order to capture the information about the system needed to support the functionalities offered by the iLAND middleware, two different views have been identified: Infrastructure and Specification.

2.1. Modeling of the Infrastructure View

The Infrastructure View contains the necessary information about system resources offered for the creation and execution of applications, from a platform independent point of view and taking into account the specificities of the platform: the services and their implementations, the nodes where the service implementations are located and the network segments through which the service implementations of an application exchange information. Figure 1 illustrates the corresponding meta-model whose elements are characterized in the following subsections.

2.1.1. Service

In the context of iLAND, a service is a software entity, with well and self-defined interfaces, that provides a specific functionality. Three types of services can be distinguished: (1) data provider services represent entities that provide information to the application, for instance a sensor, or a camera; (2) data sink services represent entities that collect information from the application, e.g. actuators, displays...; (3) application services which are in charge of executing part of the application functionality. The service interface consists of all the inputs the service requires (required interface) and all the outputs the service provides (provided interface).

The inputs and outputs are modeled by their name, description, their order regarding to other inputs or outputs, its data type and its range of values. It is also possible to have parameters that are input and output
parameters at the same time, being in this case, **inout** parameters.

### 2.1.2. Service Implementation

Each service can be implemented in different ways by several **service implementations**. They provide the same functionality, through the same interface, but having different set of QoS attributes. Two kind of QoS parameters can be associated to service implementations: (1) **Provided QoS parameters**, usually application specific, such as: compression codec used and/or the image resolution for remote surveillance applications; (2) **Required QoS parameters**, usually related to the physical resources the implementation needs, for instance its worst case execution time (WCET) on the target platform or the memory size it needs.

Service implementations can be located in the same or in different nodes. When a data provider/sink corresponds to a device, a service implementation may represent a specific operational mode of the device.

### 2.1.3. Node

**Nodes** are characterized from a platform independent point of view, and from a platform dependent point of view. The characterization of the first point of view takes MARTE [21] as a reference. Nodes are composed by a set of resources divided in five groups: (1) Memory resources; (2) Processor resources; (3) Battery resources; (4) Operating System resources and (5) Network interface resources.

### 2.1.4. Network

The nodes of iLAND applications can be located in one or more network segments. They represent the network media and certain network devices that are part of the network infrastructure such as gateways or switches. A network segment is characterized, among others, by its location, its technology (ZigBee, Bluetooth, Ethernet...), its maximum bandwidth, and the maximum number of nodes that can allocate.

### 2.2. Modeling of the Specification View

The **Specification** view allows defining iLAND applications as a service graph, i.e. a set of services connected through their input/output interfaces. Besides, it must offer mechanisms to express the application logic, including reconfiguration at run-time.

In the context of iLAND applications, a SOA is a user application that is composed by a set of services co-operating to achieve the application functionality. This cooperation is represented by a **service graph** that symbolizes how the **functional requirements** of the application are met.

Besides, a SOA can also have **non-functional requirements**. The non-functional requirements are related to different types of QoS: timing properties and activation properties such as the period if the SOA is periodic, the event name if it is event triggered or an end to end deadline. Other QoS characteristics commonly depend on the type of application.

Depending on when a SOA is created and when its necessary resources are allocated, two type of SOAs could be distinguished: (1) **Static SOAs** created from the beginning and if they are reconfigured it is by decision of the iLAND middleware, i.e. the reconfiguration is performed in an application unaware fashion; (2) **dynamic SOAs** created or destroyed at run-time by decision of the application. The former requires mechanisms to express the application functionality; and the latter also requires mechanisms for expressing dynamic reconfiguration at application level.

When a user application contains dynamic reconfiguration, the modeling of the application consists of the following sequence:

- The different SOAs have to be defined, specifying the SIS involved and the way they are connected (application logic).
- The composition of those SOAs is depicted in a reconfiguration scenario that represents the event driven SOAs creation and destruction. Note that each SOA may have associated its own QoS parameters that represent its non-functional requirements.

Figure 2 illustrates the corresponding meta-model whose elements are characterized in the following sub-sections.

### 2.2.1. Service in SOA (SIS)

A service graph consists of a set of services that are encapsulated by the so called **Service in SOA (SIS)**, as Figure 3 illustrates. SIS acts as a wrapper that defines the way in which services are connected among them, i.e. they allow composing different application using the same services.
2.2.2. Port

A SiS has one input port and/or one output port, as it is depicted in Figure 2. Ports provide access to the internal structure of the SiS, i.e. to the service that the SiS contains (for a Macro_SiS it provides access to the Macro_SiS/SiS it contains). In a sense, the SiS is an abstraction layer that allows adding connection logic among services. Thus, externally, a service receives data from other SiS through the input port, and it sends data to others SiS through the output port. Internally, input ports are in charge of executing the input connection logic for obtaining the actual input parameters of the service as well as to execute the output connection logic to obtain the data to be sent to the successor(s) from the actual output parameters of the service.

Therefore, a functional graph is formed by a set of SiS connected through their input and output ports provided with input connection logic and output connection logic, respectively.

In Summary, Input ports are the responsible for the reception of the input parameters a service needs to execute. These input parameters can be received from a unique predecessor or from a set of predecessors. On the other hand, Output ports are responsible for sending the output parameters that a service provides after its execution. These output parameters can be sent to a unique follower or to a set of followers; furthermore, this sending of data can be performed under certain conditions. As for input ports, different logic types for output ports have been identified.

2.2.3. Connector

SiS are linked by means of connectors that go from output ports to input ports, that is, they collect the information exchanged among SiS. A connector is associated to the output parameter(s) of a SiS and it is used by the input parameter(s) of a successor SiS. This information is necessary in order to correctly build the message exchanged between the two services.

2.2.4. Timeout

Taking into account that services are stateless entities, timeouts are the modeling mechanisms provided to SiS for expressing and controlling the maximum duration of a situation. The logic for the timeout activation (set upper limit and start timeout) and reset (stop timeout and reset its current value) must be based on the input/output parameters of the service.

2.2.5. Event

In order to express relationships among different SOAs, the event concept is defined. An event represents the necessity of an invocation to the middleware in order to create/destroy applications and allocate/de-allocate their corresponding resources at runtime. Events are associated to a special connector that links the event generator SiS and the SOA to be triggered. Again, the logic for the event triggering lies in output ports.

According to the necessity of iLAND applications, four different types of events have been identified so far: (1) Create event triggers a new SOA and the original SOA remains executing. Thus, after the event occurs both SOAs are executing. (2) Destroy event provokes the destruction of a running SOA (3) Replace event triggers a new SOA
and the original SOA is destroyed while the new one is created. (4) Switch event indicates that two SOAs must execute continuously and alternately.

3. Use Cases

This section presents the modeling of several use cases of iLAND applications. The iLAND middleware is being applied to three different real application domains: (1) A high-availability video surveillance application. In this case, iLAND must allow reconfiguring the functionality inside the nodes as well as the nodes themselves without interrupting the operation at any time; (2) Health-care applications. In this context, buildings are equipped with a number of sensors, actuators and distributed hosts that share information; and (3) Early-environmental detection applications through public infrastructure. These applications are based on self-organizing sensor islands that collect environmental data and share them with the outside world via opportunistic networks.

An example of a basic SOA is illustrated in Figure 4. This is a simple remote surveillance application for video monitoring; the captured images are processed and displayed in a video monitor. The SOA consists of a sequence of SiS, having each a unique predecessor service (except for the initial SiS, ImageSource) and a unique successor service (except for the final SiS, VideoMonitor). In this case, each connector has associated all the output parameters of the SiS that are used as input parameters of the successor SiS.

However, applications may demand more complex logic at its input and/or output ports. For instance, a SiS might need to send data to other SiS under a condition, or a SiS might receive data from a set of SiS or even it might receive data from one of a set of SiS or other logic conditions. Additionally, it may be desired to represent relationships among different SOAs. For example, it may be necessary to express that when a SOA is triggered upon the occurrence of an event, the activation SOA has to be destroyed. Thus, when resources are allocated to the triggered SOA, the resources corresponding to the original SOA must be released.

Figure 5 presents an example of a health care application with generic logic at an output port. The LocationSystem (LS) SiS detects that the user location has changed and sends the new location to the DailyActivityMonitor (DAM) SiS. The DomoticController (DC) SiS also sends the state of the appliances in the home to the DAM SiS. And finally the ThingsManager (TM) SiS, in charge of monitoring the use of different objects of the house (kitchen items, sofa, beds, trash can...), sends this information also to the DAM SiS. The DAM service is executed when it receives data from any of its predecessors.

The activity of the user and the home status is monitored by the DAM SiS, and when DailyReasoningEngine (DRE) SiS detects an unusual situation the user is alerted. If the DRE service detects an unusual situation, its alarm output parameter takes a TRUE value.

Another example of complex application logic is illustrated in Figure 6. It is a remote surveillance application that detects perimeter intrusion. The ImageAdapter SiS decompresses video streams (captured by an image source represented by the ImageSource SiS) and applies simple image adaptation algorithms, such as resolution changes or luminance compensation. ObjectTracker SiS extracts relevant information from the objects in movement present in these streams. The VirtualFence SiS checks if an object crosses to certain area on the image, and all this information is shown in a display (GUI SiS). The streams processed by the ObjectTracker SiS need to be stable. When an image is stable it is sent directly to the ObjectTracker service, otherwise it is previously sent to the ImageStabilizer for its stabilization.
Figure 7 illustrates a health care application that processes pulse measurements. When the ThresholdChecking SiS detects that a measurement exceeds a threshold for a patient (threshold_alarm output of ThresholdChecking service is TRUE), during a period of time previously established, medical staff is warned (WarnMedicalStaff SiS) and the patient’s glucose processing is activated creating the “Glucose Reading” SOA. Note that when the create event is triggered the new SOA is created while the previous SOA remains executing.

The logic for the event generation, for the application logic and for the timeout management is represented by the activity diagram attached to the output port of the ThresholdChecking SiS. The Start timeout_1 activity starts the timeout, this happens when the threshold_alarm output parameter of the ThresholdChecking service is true and the upper limit of the timeout has not been reached yet. If the alarm situation lasts longer than the time interval set as the upper limit of the timeout the medical staff is warned. On the contrary, when threshold_alarm output parameter of ThresholdChecking service is false, there is not alarm situation and the timeout is stopped.

4. Modeling Language

This section presents a brief description of the modeling approach by a Modeling Language.

UML, as a generic purpose modeling language, offers a start common point for describing any type of systems. Nevertheless, sometimes it is necessary to extend UML adding the particular characteristics of the system to be modeled, for example, by means of the definition of UML profiles. They consist of a set of stereotypes grouping a collection of restrictions and tagged-values. Stereotypes could be assigned to any UML element with the aim of adding the particularities of the application.

As it has been previously remarked, the infrastructure view includes resource characterization. The MARTE UML profile has been used, when possible, to characterize the nodes. As MARTE is mainly aimed to represent static properties of the nodes, operational properties have been added. In addition, iLAND requires a more detailed characterization of the network properties than the one MARTE provides. As a consequence, the UML profile for MARTE has been used when possible and new characteristics have been added in order to represent the new properties.

4.1. Infrastructure View Modeling

In order to characterize the elements of the Infrastructure, a UML profile has been defined. It consists of a set of stereotypes to apply to the UML elements of the view and a set of tagged values that characterize the features of these elements.

Services and their implementations should be modeled by UML Class elements enriched with the corresponding UML stereotypes.

The UML elements necessary for defining the hardware elements are those that can appear in UML deployment diagrams: Node elements for nodes; Device elements for memory resources, processor resources, battery resources and network interface resources; Execution Environment elements for operating system resources; Artefact elements for schedulers. Network Segments are also modeled in UML Deployment Diagrams by means of communicationPath between nodes.

4.2. Specification View Modeling

In order to characterize the elements of the Specification View, an UML profile has been defined. An UML stereotype for each element has been defined having associated a set of tagged values that represent the characteristics of those elements.

The service graph of each application is defined by a UML 2.0 component diagram. It is composed by a set of connected SiS, defining the functionality of the modeled application. Hence, each SiS in UML is defined by an UML component. As commented before, SiS encapsulates and provides application dependant logic to a service defined in the infrastructure model. In UML this link is done by means of ComponentRealization association. Input and output ports are modeled by UML port elements.
Output ports can have associated activity diagrams for application logic definition, for event generation logic or timeouts management.

The data flow exchanged between SiS is defined by connectors. In this case, the data flow itself is defined by an UML interface and the direction of the communication is done by InterfaceRealization or Usage associations.

Event concept in UML based modeling is realized by an UML dependency and the SOAs are represented by UML components.

Figure 8 illustrates the modeling of Live Video Monitoring SOA explained in section 3 (see Figure 4).

5. Conclusions

This work presents a Model Driven Approach to model composition and dynamic reconfiguration of iLAND applications. The proposed modeling is based on different view models that contain the relevant information for the interaction between the iLAND system and the system stakeholders: the user applications and the middleware components. In particular, the Infrastructure model corresponds to the characterization of the system resources offered to applications, mainly services and their implementations, nodes as resource containers and networks. This characterization is used by the middleware components in order to guarantee the application non-functional requirements are met.

On the other hand, from the application point of view, the modeling approach should offer expressive power enough as to specify the application functionality as well as application reconfiguration needs at run-time. Mechanisms to specify the functionality and execution logic of service based applications have been analyzed based on the use cases of industrial partners. Special emphasis has been devoted to specify at the modeling phase the application reconfiguration needs at run-time.

This modeling approach will be active as long as iLAND demonstrators are in design and development phase. As a result, refinement characterization of some of the model elements may be necessary and new reconfiguration scenarios may arise.

6. Acknowledgments

This work was financed by the ARTEMIS I-land Project and by UPV/EHU GIU 07/36.

7. References


