A bi-dimensional QoS model for SOA and real-time middleware

Marisol García-Valls, Pablo Basanta-Val
Department of Telematics Engineering
Universidad Carlos III de Madrid
Avda. de la Universidad 30
28911 Leganés, Madrid (Spain)
{mvalls,pbasanta}@it.uc3m.es

Marga Marcos
Departamento de Ingeniería de Sistemas y Automática
University of the Basque Country
Alda. Urquijo s/n
48013 BILBAO (SPAIN)
marga.marcos@ehu.es

Elisabet Estévez
Departamento de Ingeniería de Electrónica y Automática
Universidad de Jaén
Campus las Lagunillas S/N,
23071, Jaén, España
eestevez@ujaen.es

***Abstract*- Architecting networked embedded systems following a service-based paradigm improves the flexibility in software design and development. Services can be connected and composed to provide more complex functionality in the form of a service graph or application. Service composition must be done according to two different goals. On the one hand, service interfaces must be compatible, i.e., the provided interface of a service must match the required interface of the connecting one. On the other hand, composition criteria must meet application level requirements or QoS constrains. This paper provides an interface view of services highlighting the way in which they expose their QoS requirements in their interfaces. Based on this, it is explored how application level QoS requirements influence the specific selection of particular services. The paper identifies a bi-dimensional model of the QoS properties of services that are related to (1) the application functionality and (2) the execution requirements or computational resources needed from the platform. The paper provides a practical implementation of this QoS model in the iLAND project. iLAND middleware supports the dynamic reconfiguration of real-time service-based applications that specify their QoS requirements in this way. The paper validates the proposed QoS model with a specific use case.**

I. Introduction

As technology improves at a growing pace, new possibilities emerge that augment the functionality offered to users. However, the price to be paid is on the side of the software platforms. Software increases its complexity to support both: (1) the high degree of functionality expected by the current user and (2) the complexity of the modern hardware platforms, being their required cross-compatibility integrated in the software layers. At the level of hardware, networked embedded systems (NES) are becoming a cloud of hundreds, and even thousands, of heterogeneous nodes connected by means of heterogeneous networks as well; they are now used in various domains such as cloud or grid [32]. As a consequence, software platforms are transferred the responsibility of bridging the gap between such a deployment complexity and the user applications.

New programming paradigms and development possibilities appeared over the last decades that allow creating complex applications with a higher degree of decoupling. However, fitting the enabling run-times to some hardware characteristics is not an easy job, especially if the system under development has limitations in terms of computational capacity and real-time or QoS requirements. Therefore, complex software paradigms have to be lightened to be applied to heavily resource constrained hardware as sensor motes; and this is a very challenging problem. Moreover, QoS requirements have to be expressed in an efficient way inside selected software paradigms so that specific support for the different execution platforms can be inferred.

In this context, the *service-oriented architecture* (SOAs) paradigm presents itself as a design and development model of high interest for two main reasons: (1) it decomposes system functionality into functional units allowing to implement decoupled systems, and (2) it defines a simple and flexible interaction model based on message exchange. Services are self contained functional pieces with well-defined interfaces that contain their functional and non-functional specifications. Interfaces are of two types: *provided* to and *required* from other services. By appropriate composition of services, groups of services that provide extended functionality can be created, and these are named *applications*. Preserving the provided and required interfaces, replacement of services is possible, and this is the basis for application reconfiguration.

NES have different levels of temporal requirements, i.e, they can be hard or soft real-time. In soft real-time NES, the concept of QoS-based execution applies referring to trading off the assigned computational resources for the quality of the delivered output. Real-time NES have to preserve their temporal properties both during normal operation and also in the event of a reconfiguration or a transition to a different mode of operation.

The execution platform must provide some additional logic to guarantee predictable behavior. This logic usually comes in the form of an additional software layer/module/component. In the iLAND approach, this logic is integrated in the middleware that is a real-time communications middleware and a reconfiguration arbiter that manages and coordinates the transition of applications from one configuration to another. For arbitration of service execution, the middleware needs information about the characteristics of the running services and applications specially their resource requirements. Applications and services are modeled containing a set of parameters that capture the set of properties that are needed in order to define their behavior and requirements; this way, the middleware can manage their execution accordingly, meeting their execution needs. In this paper, we refer to these parameters as the QoS properties. It is, then, needed to efficiently identify and characterize the QoS properties of applications and services.

In NES, research work has mainly focused on the generalities of QoS characterization with the main focus on the non-functional properties related to the requirements for computational resources; the majority of approaches have not dealt with the separation of application semantics related to the specific data they process from the pure computational requirements. In a previous work [25], the authors described an initial QoS characterization for services and provided a component model based on a simple template for iLAND services to have a homogeneous implementation. Such a component model has been used in the context of iLAND project [26][17] aimed at developing a middleware architecture for time-deterministic composition and reconfiguration of service-based applications. For achieving real-time reconfiguration of service-based applications, there are some important questions to be addressed. First, the reconfiguration model of service-based applications must be settled, as well as the service characterization and definition. Secondly, algorithms for composition of services must be developed that can be executed in a time-deterministic way. Moreover, the middleware plays a key role in this scene, since it will coordinate the execution of the applications/services and will coordinate their composition and reconfiguration. Eventually, real-time (or at least, quality of service –QoS) execution support and resource management schemes [7][8][6][5] should be offered by the operating system and the network. The current paper does not focus on the middleware architecture and its specific algorithms for enabling service composition and reconfiguration; these have already been addressed in previous contributions [27][28][29]. Instead, authors enhance the initial work on the characterization of services with QoS parameters used in iLAND [25] with the contributions of the project for modeling of the SOA-based applications and the identification of a comprehensive characterization of the QoS properties of services. The paper establishes the connections between the QoS properties of applications, and that of services. Moreover, it also defines the types of service graphs that the iLAND middleware manages to represent applications and to derive new application. QoS properties are developed in two dimensions: those related to specific data nature of applications and those related to the physical resources.

In section 2, this paper presents the related and previous work on characterization of QoS properties of real-time NES in software oriented environments. In section 3, the service-based model for QoS-sensitive applications is presented together with the graph types that model the structure of applications; this section also contains the description of the internal structure of services and applications. Section 4 presents the bi-dimensional QoS characterization details and it puts in relation the properties of both applications and services. Section 5 presents the validation of the concepts and the implications of this characterization in the composition of services on an example application (use case) setting. Section 6 concludes the paper.

II. Related work

The originator of most of QoS theory mainly for probabilistic packet transmission has been the networking area. In it, QoS properties are directly mapped into packet tagging or network reservation as DiffServ[30] and IntServ [31][31], respectively. Other network-related work has been mostly carried out over these ideas.

Modeling the QoS properties of software systems has been mainly addressed from the resource management and component model side. Proof of this have been the different profiles defined by organizations as OMG on the definition of resource management related properties of systems. Perhaps the most influencing work on QoS characterization have been the OMG standards for UML[1] and its modeling of QoS and Fault Tolerance [3] and the profile for schedulability, performance and time (UML SPT) , and, more recently MARTE with numerous contributions and extensions for different domains such as [35][36]; they are oriented to the construction of models that can be cross-checked for assessing non-functional properties facilitating communication of design ideas at development time. Based on these works, component technology work has also been proposed for extending complex CORBA-based component models[20][21][22].

Besides this activity on standardization, different scientific contributions have provided architectures for resource management including a comprehensive characterization of QoS properties related to real-time applications and real-time tasks for embedded systems. One of the main approaches in this direction was the HOLA-QoS architecture [4] that included a set of protocols for mode changing [5], dynamic priority assignment [6] (recently extended for dual priority bands[7] and mobile devices [10]), and other contributions for specific application domains such as ambient intelligence [8] . These ideas were also later mapped to the services paradigm precisely for the Jini based environments as CoSeRT [9]. Initial distribution models being designed also as [34].

QoS modeling for n-dimensions was already explored by [24] where a fairly comprehensive characterization of QoS properties was made but still with many limitations due to its purely theoretical perspective that lacked any application semantics; moreover, the final goal of composition was also not the driving force. Also, previous contributions on component based systems as [20][21][22] proposed efficient ways to integrate QoS support in actual software execution frameworks. More recently, QoS properties have also been combined with real-time replacement models to support dynamic real-time systems [23]. This is especially beneficial for domains like real-time media processing and rendering [16][15] developed with the aid of middleware such as in [11][13][14][17] or media lab infrastructures [18]. Also, mobile operating systems can be potentially benefited from improved resource management [10].

The SOA community has also addressed the characterization of QoS properties of services and applications. However, they have addressed almost exclusively web services, and these are a specific class of services that do not have real-time concerns. Therefore, work in that area has focused at cross-checking and composition of web services based on their functional descriptions in the form of ontologies. A few approaches to composition algorithms specifically for real-time systems has appeared; the initial approach for distributed real-time systems was presented in [9] as the germ of this idea.

The work currently presented in this paper focuses at a pragmatic service characterization for QoS properties to enable the development of a system based on services that can be managed by the middleware; the middleware is capable of adjusting to the needs of the environment (environment triggers, user requests, and even internal monitoring decision as resource usage/availability needs, etc.). This characterization is used in the context of iLAND project. An initial approach to this was presented by the authors in [25] where the objective was to describe a component model for homogeneous service implementation. The current work extends it significantly by defining the possible application graph structures, and by enhancing the characterization and modeling of services and adding the description for application-level QoS; moreover, the relation among both application and service QoS properties is also presented. Also, implications on the composition logic is added and its relation to the composition logic of the middleware is indicated.

III. Service-oriented System Model

Service entities are the building block of *service oriented architectures* (SOA). A service is a piece of functionality that receives and sends messages in a remote or centralized environment communicating with other services via message exchanges. Service oriented computing allows building distributed applications in a decoupled way where services reside in remote nodes in the network and communicate via messages or events. In our approach, a service manages a data channel as shown in figure 1. It receives data through some input interface, and it processes the data generating a result that is delivered to other services through its output interface.

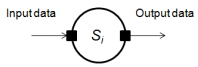

Fig.1. Data channel flowing through services. Services process input data (messages) and generate output data (messages)

It is possible to build extended functionality (applications) by connecting services. *Service-based applications* are a set of connected services in the form of a graph. An application is a graph-based structure where the *nodes* are the services and the *connecting lines* or arrows are the messages exchanged between them. Figure 2 shows the scheme of this idea.

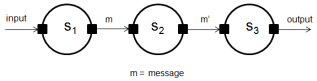

Fig.2. Service-based application has a graph structure

In the proposed model, a service is a self contained execution unit that is mainly characterized by the functionality it offers. Such functionality is its *provided interface* that integrates the set of operations and data that the service can execute. In the same way, a service has a *required interface* with the set of operations and data that it can take as input. Therefore, a service implementation is a particular version of a specific service. The set of all services of a system is given. Following, the set of all services in the system, S, is defined as $S = \{s_i;\ i = 1,..,n\}$ where n is the number of services in the system. A specific application a_j is a subset of S, in the form $a_j \subseteq S$.

A specific service s_i is realised or implemented by one or more *service implementations* that are the actual running entities. Therefore a specific service s_i has a number of service implementations, $s_{i,l}$, in the form $\{s_{i,l};\ l = 1,..,m\}$. The set of all service implementations in the system is $SI = \{s_{i,l};\ i = 1,..,n; l=1,...m\ \}$ where n is the number of services in the system and m is the maximum number of service implementations per service. The number of service implementations per service can vary between different services. The service implementations of a single service are called *companions*.

As a consequence, a running application is a collection of *service implementations*. In a similar way, an application actually contains only service implementations. In this model, a system is described by three types of graphs are shown below:

- *Application graph* (AG_i) of application i is a graph structure that contains only services and showing the relations among them. Therefore, it reflects only the functionality, and it is the static view of a given application.

  $AG_i = \{ S, R, Q \}$ where S_i is the set of services of application i; R is the set of relations (arrows) between nodes and contains elements of type $S_j \rightarrow S_k$ where service S_j is connected to S_k in the graph; Q is the set of quality of service parameters (related to the specific application data processing needs and its computational resources requirements that is detailed in section IV.

- *Expanded graph* (XG_i) is derived from substituting each service in the AG_i by its set of service implementations. XG_i shows the running entities of an application. This is the graph to search through by the composition algorithm.

  $XG_i = \{ SI_i, R', Q \}$ where SI_i is a set of service implementations of application i $(SI_i = \{s_{i,l};\ i = 1,..,n;\ l = 1,..,m\})$; R' is the set of relations (arrows) between nodes/service implementations and contains elements of type $S_{j,x} \rightarrow S_{l,y}$ where service implementation $S_{j,x}$ is connected to $S_{l,y}$ in the graph. This relation implies that $S_{j,x}$ sends messages to $S_{l,y}$ and that the provided interface of $S_{j,x}$ is compatible with the required interface of $S_{l,y}$. Q is the same set of quality of service parameters of its application graph.

- *Execution graph* (EG_i) is the application to be executed therefore including only the active service implementations.

  $EG_i = \{ SI_i, R'', Q \}$ where SI_i is a set of service implementations of application i and only the ones that have been selected by the composition logic; in the same way, R'' and Q have the same structure as for the XG^a. R'' is a subset of the relations contained in R': $R'' \subseteq R'$.

Below, an exemplification of the graph structures that model a service-based application in the iLAND approach is provided. Initially, the set of services of a given application have to be provided in the form of an application graph, AG. Figure 3 presents an *application graph* made of three services.

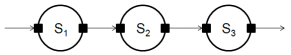

Fig.3. System represented by its application graph (AG)

Figure 4 presents the expanded service graph resulting from the substitution of each service from the AG by its set of service implementations. In this case, service s_2 has three implementations (or companions) contained in the set $\{s_{2,1}, s_{2,2}, s_{2,3}\}$.

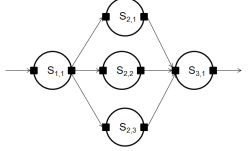

Fig. 4. Expanded application-graph where service s_2 has three companions

Therefore, the expanded application graph shows the different possibilities for deploying a real application. At the end, a selection of service implementations will have to be performed to obtain the actual execution graph as shown in figure 5. In this case, $s_{2,3}$ have been selected for s_2.

Fig. 5. Execution graph; the composition algorithm has selected $s_{2,2}$ to execute in representation of service s_2

SOA based systems adjust naturally to the software reconfiguration model of iLAND. Application reconfiguration is achieved by replacing at least one current service implementations. Events that trigger reconfigurations can be broadly classified in two broad groups:

- *User action*; a request may be issued for either a change of a functionality of an application or a change in the output quality delivered by some application which requires a different companion to achieve it.
- *Internal system decision* detected by the internal execution monitoring of the system itself. For example, the system can detect if the processor load is very high and load balancing mechanisms need to be launched to preserve the level of resource assignment and, therefore, the application execution quality.

The above triggers lead to two fundamental types of reconfiguration:

- *Functional*; a service has to be replaced by a different service or void, i.e, simply stopped. A transition from figure 5 to 6 is a functional reconfiguration since $s_{2,2}$ is stopped.
- *Internal*, a service implementation is replaced by another companion.

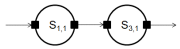

Fig 6. Functional reconfiguration; taking figure 5 as starting point, service $s_{2,2}$ is removed

Fig. 7. Internal reconfiguration; taking figure 5 as starting point, the application replaces $s_{2,2}$ of figure 5 by a companion

IV. Bi-dimensional characterization of QoS properties in iLAND

A comprehensive characterization of SOAs has been done in iLAND project on the basis of the simple model presented in the current paper. QoS characterization has been achieved both at service and application levels. Application-level QoS parameters describe the end-to-end expected behavior, whereas service-level QoS parameters describe the local characteristics of specific services. Application-level QoS parameters directly influence the service implementations that will be elected to run.

A. Service-level characterization

Services can be connected in an application graph; therefore, they must explicitly provide connection hooks or ports to interact with other services of the application. Following, the characterization of a generic service is given.

```
Service template Service_Gen(functionality: string){
  Description = functionality;
  Input_interface = input_port_1;
  Output_interface = output_port_1;
  Sid: generate_unique_id(functionality);
}
```

The definition of a service contains a description of the specific functionality it provides; that is the primary search key to find a service. Each service also contains a unique identifier in the system. Moreover, to interact with other services, it must specify its input and output interfaces named `Input_interface` and `Output_interface`. Data and event exchange is carried out through these ports. These elements describe the functionality that the service provides to (and requires from) the outside world, or to other services, must be also part of its definition:

- *Required or input interface* integrates the set of expected input data, events, and information as well as the operations provided by the service. Such operations receive data or parameters to process them for generating a result. Results will be sent to other services via the output interface. Other services can use the required interface for communication with this service in a push mode.
- *Provided or output interface* includes the results generated by the service. These data will be the incoming information for other services. Composition algorithms will make use of such interface to determine compatibility information indicating whether the service can be composed with others. Also, other services will issue communication requests to it through this interface.

Following, the description of a service implementation is given.

```
ServiceImpl template IM_Gen(im_id: Id, sid: Id,
qos_p: Resources, node: Localization, code: Function ){
  Id: im_id;
  Sid: sid;
  Qos_Params: qos_p;
  Localization: node_id;
  Function: code;
}
```

Each *service implementation* is a particular realization of a service. Since it is an active entity that executes the functionality of the service, it requires to consume computational resources. The information regarding its resource needs is important for the composition algorithms; schedulability analysis is part of the composition algorithms to determine a feasible execution set.

B. Application-level characterization

Application requirements are end-to-end properties of the service graph. An application (as shown in figure 3) is, therefore, characterized in the following way:

```
Application template App_Graph(){
  Description = functionality;
  ServiceSet = service_list;
  QoS_Params = qos_p;
  Aid: generate_unique_id(functionality);
  }
```

The `QoS_Params` contain the information related to the characteristics of the global functionality of the application. They characterize the data flow through the whole application and model its end-to-end requirements. These information include the characterization of the *data* that the application processes and lower-level values related to requirements for *physical resources* that are the needs for computational resources (i.e., processor cycles, memory, battery, network bandwidth, etc.). The `ServiceSet` attribute contains the list of graphs that the system requires to manage the execution of applications. As explained above, each application *i* has an AG_i *(application graph)* that contains only the service structure, an XG_i (*expanded graph*) generated by the composition algorithm, and the selected EG (*execution graph*) wich is the set of service implementations that are running.

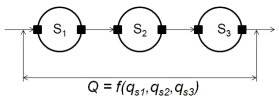

Fig. 8. Application data channel flows through services'data channels

If the application defines that its input data is of type d_x and its output data of type d_y, this requirement influences directly the input and output data parameterization of the elected services and service implementations. Therefore, it is easily observable that $EG_i = f(Q_i)$ meaning that the selection of the running services depends or is a function of the application level QoS parameters (Q), where $Q_i$is the set of characterizing QoS parameters of application i, i.e., the end-to-end parameters of the whole application data flow as shown in figure 8. Conversely, Q must be compatible with the QoS parameters of the specific service implementations of EG_i that are indicated by q_{s1}, q_{s2}, and q_{s3} in the specific example of figure 8.

C. *Identifying the bi-dimensional aspect of QoS properties in real-time SOAs*

One of the main contributions of the iLAND model is that both at the level of applications and service implementations, *QoS characterization* has *two dimensions*. Applications and service implementations have requirements for: (1) physical resources that provide the computational means and (2) requirements of the processed data.

It is then considered that services receive two types of information through the required interface related to the data that the application processes it. On one side, *data qos* that relate to the characteristics of the functional information handled by the application. For example, an audio service will receive audio samples that is part of the functional description of the service. Also, it will receive information on the QoS of the audio samples since the quality of the processed data depends, in this case, on the bits/sample, noise-related information, and even the required computation resources (as processor and memory needs). This second part are the *data QoS* parameters. Access ports of figure 9 present this idea in which a generic service component receives both types of information that is combined inside the service; also, the relation to an audio service is shown.

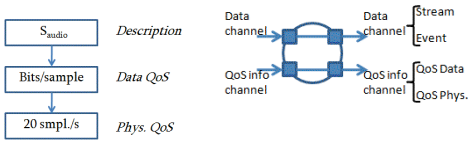

Figure 9. Relation of QoS parameters for describing data and physical resources

The proposed charaterization provides a bi-dimensional QoS specification model identifying two types of QoS properties, those related to the application data (*data-related)* and those related to the required computational resources (*physical-resources QoS information)*. The nature of the processed data imposes non functional restrictions on real-time applications that have an immediate translation to requirements for computational resources. Since the audio service of figure 9 requires to processes 20 samples/s., it has an output deadline of 50 ms. This translates to a specific processor usage requirement periodically, i.e, every 50 ms the audio service has to execute for a given amount of time.

This concept is implemented in service implementations since their model inherits the ports of services; service implementations have input and output ports to propagate their information of QoS for both dimensions as shown below (input and output ports are of type `QoS_Port`):

```
QoS_Port template QInOutPort(q_data: QoS_AppData, q_phys:
QoS_PhysResources) {
  QoS_Data: q_data;
  QoS_PhysResources: q_phys;
}
```

This provides a unified interface that integrates the information received and output by a service component. Data delivered by a service may be the input to another service in the application chain of services. Such information is provided together with the QoS parameters that define it in an integrated way. This is an enabling mechanism for propagating the quality of service parameters along the application chain, so that especially data dependencies can be accounted for.

D. *Relating Application and Service Implementation QoS parameters*

QoS information must be specified also for applications. Therefore, *service parameters* and *application parameters* have a direct relationship that is shown in figure 10. Application QoS parameters are enforced for all the application service graph. To execute an application, the system will select a set of service implementations in such a way that the application-level QoS parameters are fulfilled.

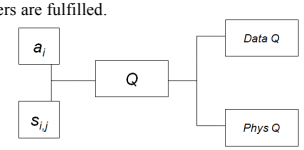

Fig. 10. QoS info (Q) is contained in both applications (a_i) and service implementations ($s_{i,j}$). Q presents two dimensions: application-level data and information on the requirements for computational resources

Application level QoS parameters are also related to the physical resources in a direct way, as presented in figure 10. The higher the QoS offered by the application, the more computational resources it requires. Multimedia applications are a typical case of this since multimedia tasks are greedy resource consumers, and the quality of the output they produce is in direct relation to the amount of physical resources that is assigned to them.

Therefore, QoS parameters of an application are also split in two dimensions according to their relation to:

- *Application-level perception (Data Q)*: these parameters refer to the specific functionality of the application such as input and output data characteristics that map to the output results of the service. They have direct impact on the level of satisfaction of the user expectations.
- *Needs for physical resources(Phys Q);* these parameters are tipically processor utilization (CPU time, period, deadline, and priority), memory usage, network bandwidth, or battery consumption.

Application-level QoS related to physical resources are the following:

- End-to-end deadline. The deadline may coincide with the end-to-end period if all tasks are modelled as a pipeline where incoming data enters periodically.
- End-to-end memory. This corresponds to the total amount of main memory assigned to all application service implementations.
- End-to-end battery/power. Energy management is related to the amount of CPU cycles executed and the usage of the network resource, especially for embedded motes in a wireless environment.

Since service implementations are part of the execution graph of applications, QoS parameters of service implementations and applications (as shown in figures 8 through 10) exhibit dependencies. In this way, both templates, `QoS_InPort` and `QoS_OutPort`, are applied to both, applications and service implementations.

Property	Value
Service functionality:	Media processing
Media type:	Audio samples
Data QoS:	# bits / sample
Phys QoS:	20 samples / sec (T = 50 ms; D = 50 ms)

Fig. 11. Bi-dimensional QoS characterisation of an example audio processing application

As a result of the bi-dimensional QoS characterization of properties, the application graph describes a virtual channel that connects all service implementations input/required and output/provided ports (see figure 11 and 12). *Data QoS* properties flow across service implementations that have compatible input and output ports. *Phys QoS* properties are used by the systems (i.e., the middleware) to determine the feasibility or real-time schedulability of the whole application at run-time. The latter is done by applying an admission protocol based on schedulabitity analysis.

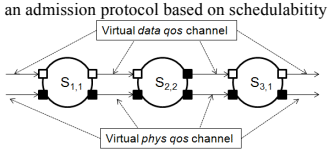

Fig. 12. Application-level channels for data-related and QoS-related information.

To compose a given application, the selected service implementation set must fulfill that for all services connected, their *required data QoS* ($idq_{i,k}$) is compatible with the *provided data QoS* ($odq_{i,k}$) of the preceding service implementation in the pipeline. This is shown in figure 13.

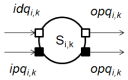

Fig. 13. Connection compatibility at the time of composition.

The selected set of service implementations must match:

- Firstly, the *application-level data-related parameters*; only the service implementations that process the required data characteristics are of utility to the execution graph. For example, if a video application is to be constructed, only services that process video data will be selected.
- Secondly, the limitations imposed by *the application-level physical resources* (end-to-end parameters) and the availability of resources in each node.

The selected set of service implementations that matches the application end-to-end data-related parameters is obtained as a function of the values of the application-level data parameters (*DQ*) represented by $\{idq_{i,k}, ..., idq_{s,k+m}\} = fd(DQ)$. There is a match in the service chain so that output of a service is compatible with the input of the consecutive service in the graph: $idq_{i,k} = odq_{s,k-1}$, being the equality operator a relation of compatibility. This function (*fd* or *function of the selected data types*) is not a utility function. *fd* selects a specific service set that is able to process the required data requested by the application; it is an ontology-based function that performs cross-check of functionality tags. Once this function is applied, a set of data-compliant service implementations is obtained, $DSI=\{idq_{i,1}, ..., idq_{s,k}\}$. This set defines the expanded graph of the application, XG_x.

The composition algorithms are applied to this set, *DSI*, to obtain the set of service implementations that adjust to the application-level (end-to-end) physical-related QoS. Therefore, the selected execution graph of an application, EG_x, is the result of a function (*fPR*) that applies a search of service-implementations based on the resource consumption of individual service implementations and of the application as a whole: $EG_x = fPR(DSI)$.

If an application end to end QoS value D_x is the global response time, then the following should be observed: $D_x \leq max(D_{i,j})$, where $D_{i,j}$ is the deadline of the service implementation that is in the final position in the application pipeline graph, $s_{i,j}$. Figures 13 and 14 illustrate the relation between the QoS parameters of applications and service implementations.

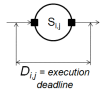

Fig.14. Physical QoS parameter ($D_{i,j}$) for a service implementation $S_{i,j}$

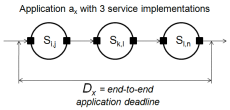

Fig. 15. Physical QoS parameter (D_x) for an application a_x

Response time analysis can be applied to the whole service implementation graph to obtain a feasible set of service implementations. Because service implementations execute concurrently with others, this model should be extended to include response-time values such that $R_x \leq \sum R_{i,j}$ for all $s_{i,j}$ of the application a_x. R_x is the response time of the whole application output and $R_{i,j}$ is the response time of service implementations.

The physical QoS parameters are defined below as the requirements for computational resources:

```
QoS_PhysResources template QoS_R(c: time, t: time, d:
time, p: priority, bat: battery, n: network_bw){
  cpu_t = c;
  period = t;
  deadline = d;
  priority = p;
  power = bat;
  n = network(transm_c, transm_t);
}
```

QoS parameters related to the requirements for physical resources encompass the different resources of a given hardware computation device including the network. The network bandwidth is the consumed percentage that the transmission requires.

The specification of a service component must include a separation between the two QoS dimensions. Following, QoS parameters related to the nature of the application data are described:

```
QoS_Data template QoS_D(){
   QoSData = data_q_set;
}
```

Data-related QoS parameters are completely dependent on the application nature and the characteristics of the data it processes. For instance, in a multimedia application, data QoS will comprise attributes as: resolution, bit-rate, and image size, among others, and in an audio application the exemplification of this two dimensions has been put forward in figure 11

QoS parameters are common for applications and services. The composition algorithm is the entity that considers their values to obtain the appropriate selection of service implementations in such a way that the combination of QoS parameters of service implementations do not conflict with the accomplishment of application-level QoS requirements.

The provided bi-dimensional characterization of the QoS properties for real-time SOA based applications is a simple model that has been integrated in the iLAND middleware. iLAND provides a modular architecture based on the concept of extended schedulability guarantees born in [4], later elaborated for services in [9] and finally implemented for real-time distributed domains in [26] following a real-time reconfiguration model and scheme.

The overview and utility of iLAND is scketched in figure 14. The main functionality of iLAND is enumerated in it, and it consists of providing an execution infrastructure for enabling QoS-based execution of services that require to communicate and execute preserving their specific temporal requirements; the most novel part of iLAND consists of the built in logic for detecting reconfiguration events and managing the reconfiguration of applications in real-time. Figure 14 shows the execution in different nodes. Temporal schedulability of distributed systems is integrated in the middleware to deliver guaranteed service execution.

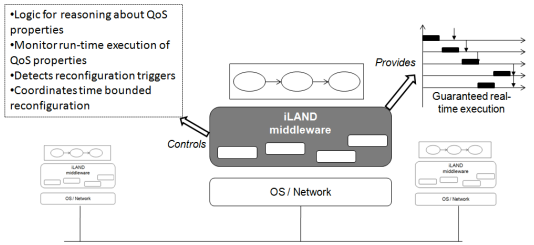

Fig. 14. Overview of the functionality of iLAND middleware

Figure 15 presents the specific integration of the QoS characterization model with iLAND.

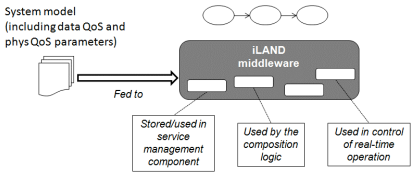

Fig. 15. Interaction of iLAND logic with the bi-dimensional QoS model

The operation of iLAND middleware refers constantly to the QoS model of the systems, since the execution of services and applications requires to be time-bounded. The model of the system indicates the number of services, service implementations, and all their required properties. Initially, the QoS model is stored in the internal middleware database. It is from there on used for all operations concerning detection of reconfiguration triggers, configuration of a new applications (i.e. selection of a new service implementation set), and enforcement of the real-time properties of all operations.

V. Model-based application of the Bi-Dimendional QoS model

The characterization of distributed and reconfigurable applications that exhibit a growing complexity is currently changing from code centric to a model centric approach, as they can benefit from meta-modeling techniques. A model centric approach relies on the use of models to represent the domain elements and their relationships. These models act as the input and the output at all stages of the development cycle until the final system is itself generated [33].

This section presents an ontological meta-model for iLAND applications, taking into account that it must capture all the information needed to run the application under the control of the iLAND middleware. The elements commented in the previous sections, their characterization in terms of QoS attributes and their relationships can be expressed as a meta-model. Fig 16 illustrates the proposed meta-model using a UML class diagram.

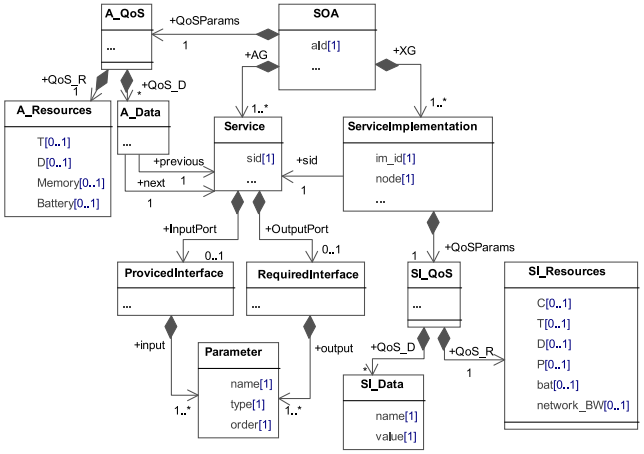

Fig. 16:QoS Bi-dimensional meta-model for iLAND service oriented applications

A Service oriented applications (SOA) as understood in iLAND, is characterized by an identifier (*aId*). Its functional requirements are expressed as a service graph (*AG*), that has to be registered in the middleware using the API. Services are characterized by the provided and required interfaces, that group a set of parameters. A parameter is characterized by the *name*, *type* and *order* properties. The logic of the application expressing the functional requirements is modeled in terms of data QoS (*QoS_D*) element characterized by the *previous (precedent service)* and *next (subsequent service)* properties. As illustrated in Fig.16, the value of those properties is a service's identifier. Hence, an initial service cannot have the *previous* property as it does not have a precedent service and a final service cannot have the *next* property as it does not have a subsequent service. The Qos Physical parameters mentioned above are grouped in Qos_R element, which can be characterized by end-to-end period (*T*), end-to-end deadline (*D*), end-to-end *Memory* and *Battery* properties.

The eXpanded Graph (*XG*) must consider the set of service implementations corresponding to every service participating in the SOA application graph. Every service implementation is characterized by and identifier (*im_id*), the id of the service it implements as well as the *node* where it is deployed. As commented above, the functional QoS parameters are very related to the application field. Thus, a couple of generic properties (*name* and *value*) have been defined. These properties have to be defined for the application field. Finally, the physical QoS parameters associated to service implementations, *QoS:R*, are characterized as a set of properties: computation time (*C*), period (*T*), deadline (*D*), Priority (*P*), battery (*bat*) and network bandwidth (*network_BW*) .

VI. Validation

The validation of the QoS model has been carried out on a simple scale video surveillance application for remote monitoring of industrial processes, implemented and deployed in a distributed environment [26]. In particular the demonstrator consists of a full-HD video application delivering a frame rate of 25 frames per second. The application graph is composed by three services: (1) video capture, that manages the camera to obtain an image every 40ms; (2) video compression whose computational time should not be greater than 40ms, in order to assure that there is no loss of images; and (3) video display which displays images on the screen that must have the same maximum computational time. This system generates alarms simulating that the monitoring parameters of the physical process has to be changed e.g. connection/disconnection of camera devices or change of image compression formats.

The following table details the characterization of the three application services, following the meta-model presented in the previous section:

Table 1: Services of video surveillance application

	Video capture	Video compression	Video display
sid	Camera	Compression	Display
Required interface	----	*IplImage* Image	*IplImage* CompressedImage
Provided interface	*IplImage* Image	*IplImage* CompressedImage	----

Regarding the QoS parameters of the whole application, they are listed in Table 2. Two QoS Data defines the logic of the application. Services that do not appear in a *next* field are considered as initial services. In the same way, services that do not appear in a previous field correspond to final services. The second QoS parameter, related to the computational resources needed to execute the application, is formed by two parameters: end-to-end period (40ms) and end-to-end deadline (120ms).

Table 2: QoS Parameters for Video surveillance application

	Video surveillance application's QoS	
Qos Data	1	Previous= *Camera*; next= *Compression*
	2	Previous= *Compression*; next= *Display*
Qos Resources	T=40ms	
	D=120ms	

Each service of the application may have a set of different service implementations deployed in the nodes of the system. In this case, it is assumed that the following ones have been registered:

As the platform is composed by two cameras *Guppy F-080C* and *GX1050C Proxilica*, there are two service implementations, one for each, having both different resource requirements, as illustrated in Table 3. Again, it shows the characterization of both service implementations following the proposed meta-model. Concerning QoS Data parameters, vision applications must consider at least two QoS data: *resolution* and *frames per second*. Physical QoS parameters are: period, deadline and computation time. As GX1050 works with higher resolution than Guppy a higher computation time is derived.

Table 3: Service implementation for Camera

	Guppy F-80C		GX1050 Proxilica	
im_id	*Guppy*		*GX1050*	
sid	*Camera*		*Camera*	
node	*Node_1*		*Node_2*	
QoS Data	name= *resolution*	value= *640x480*	name: *resolution*:	value:*1024x1024*
	name=*fps*	value=*25*	name=*fps*	value=*25*
Qos Resources	T=*40ms*		T=*40ms*	
	D=*40ms*		D=40*ms*	
	C=*20*ms		C=*40ms*	

On the other hand, two service implementations for the *Compression* service have been defined. Compressed files are significantly smaller than their uncompressed counterparts, and they fall into two general categories: "lossy" and "lossless." Lossless compression ensures that the complete image information is preserved. Lossy compression, by contrast, can create file sizes that are significantly smaller, but achieves this by selectively discarding image data. In concrete, *TIFF* compression service implementation has been developed as lossless compression and JPG service implementation as lossy compression. The QoS data of such service implementation is the compression algorithm: LZW and JPG respectively. These service implementations have the same QoS physical parameters as the Camera's service implementations.

VII. Conclusions

This paper has presented the characterization of applications based on services specifically concerning their QoS properties. In real-time service based applications it is important to capture both the functional dimension as well as the execution dimension; at run-time, the first one is influenced by the latter one. Based on previous works on the iLAND project where a component model for homogeneous service programming was introduced, this work has enhanced the description with a presentation of the bi-dimensional aspect of the QoS properties relating application-level properties to the service implementation ones. Moreover, the integration of the bi-dimensional QoS model with the iLAND middleware has been presented. A model based design of a representative use case has been elaborated to validate the easy of use and application of the model in a real sample scenario that has been proposed for iLAND, where physical resource mappings are observed and enforced in the underlying execution platform based on the provided system model.

Acknowledgemets

This work has been partly supported by the iLAND project (ARTEMIS-JU 100026) funded by the ARTEMIS JTU and the Spanish Ministry of Industry, Commerce, and Tourism, ARTISTDesign NoE (IST-2007-214373) of the EU 7th Framework Programme, and by the Spanish regional project REM4VSS (TIN 2011-28339), and eMadrid (S2009/TIC-1650).

References

[1] OMG. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, vol. 15, 2009.
[2] OMG. "Quality of Service for CORBA Components". OMG Document ptc/2007-08-113. 2007.
[3] OMG. "UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms. Version 1.1." April 2008.
[4] M. García Valls, A. Alonso Muñoz, J. Ruíz Martínez, A. Groba, "A An Architecture of a QoS Resource Manager for Flexible Multimedia Embedded Systems". In Proc. of 3rd International Workshop on Software Engineering and Middleware (SEM2002). In Lecture Notes in Computer Science vol. 2596. 2003.
[5] M. García Valls, A. Alonso Muñoz, J.A. de la Puente. "Mode Change Protocols for Predictable Resource Management in Embedded Multimedia Systems". In Proc. of 6th IEEE Conference on Embedded Software and Systems. Hanzhou, Zeijan, China. May 2009.
[6] M. García-Valls, P. Basanta-Val, I. Estévez-Ayres. "Dynamic priority assignment scheme for contract-based QoS Resource Management". In Proc. of 7th IEEE Conference on Embedded Software and Systems. Bradford, UK. June 29th - July 1st, 2010.
[7] M. García Valls, A. Alonso-Muñoz, J.A. de la Puente Alfaro. "Dual band priority assignment mechanism for dynamic QoS resource management." Future Generation Computer Systems vol. 28(6), pp. 902-912. June 2012.
[8] C. Otero, L. Steffens, P. van der Stok, S. van Loo, A. Alonso, J. Ruiz, R. Bril, M. García Valls. "Ambient Intelligence: Impact on Embedded Systems Design". *Chapter On* QoS-Based Resource Management for Ambient Intelligence. Kluwer Academic Publishers. 2003.
[9] M. García-Valls, I. Estévez-Ayres, P. Basanta-Val, C. Delgado-Kloos. "CoSeRT: A framework for composing service-based real-time applications". In Business Process Management Workshops, Lecture Notes in Computer Science, vol. 3812, pp. 329-341. 2006.
[10] M. García-Valls, A. Crespo, and J. Vila. "Resource management for mobile operating systems based on the Active Object model". International Journal on Computer Systems Science and Engineering (accepted for publication). November 2012.
[11] D. Xu, K. Nahrstedt, D. Wichadakul.. "QoS and Contention-Aware Multi-Resource Reservation". Cluster Computing, vol. 4 (2), pp. 95-107. ISSN 1386-7857. Kluwer Academic Publishers. 2001.
[12] M. García-Valls, A. Alonso, J. A. de la Puente. "Dynamic adaptation mechanisms in multimedia embedded systems". In Proc. of 7th IEEE International Conference on Industrial Informatics, pp. 188-193. Cardiff, UK. June 2009.
[13] L. Nogueira, L. M. Pinho. "Time-bounded distributed QoS-aware service configuration in heterogeneous cooperative environments". Journal of Parallel and Distributed Computing, vol 69 (6), pp. 491-507. June 2009.
[14] C. Gill, J. Gosset, D. Corman, J. Loyall, R. Schantz, M. Atighetchi, and D. Schmidt. "Integrated adaptive QoS management in middlware: A case study". Real-Time Systems, vol. 29 no. 2-3, pp. 101-130. March 2005.
[15] B. Bouras and A. Gkamas. "Multimedia transmission with adaptive QoS based on real-time protocols". Journal of Communication Systems, no. 16, pp- 225-248. 2003.
[16] M. García-Valls, P. Basanta-Val, I. Estévez-Ayres. "Adaptive real-time video transmission over DDS". In 8th IEEE Conference on Industrial Informatics. Osaka, Japan. July 2010.
[17] iLAND project. "iLAND Reference Implementation Installation & User Guide". http://sourceforge.net/projects/iland-project/ Available on-line. September 2012.
[18] M. García-Valls, P. Basanta-Val. "Usage of DDS Data-Centric Middleware for Remote Monitoring and Control Laboratories". Transactions on Industrial Informatics, vol. 9(1), pp. 567-574. February 2013.
[19] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari. "AQuoSA - adaptive quality of service architecture". Software - Practice and Experience, vol. 39; 1-31. 2009.
[20] M. A. de Miguel et a. "QoS-Aware Component Frameworks". 10th IEEE Int'l Workshop on Quality of Service, pp. 161-169. 2002.
[21] M. Shankar, M. de Miguel, and J. Liu. "An End-to-End QoS Management Architecture". In Proc. of the 5th IEEE Real-Time Technology and Applications Symposium (RTAS 99). IEEE Computer Society. 1999.
[22] M.A. de Miguel and T. Toledano. "QoS Specification for QoS-Aware Components". In Proc. of the 30th Euromicro Conference (ECBSE 2004). IEEE Computer Society. September 2004.
[23] J. Cano Romero, M. García Valls. "Scheduling component replacement for timely execution in dynamic systems". Software practice and experience, Wiley. DOI: 10.1002/spe.2181. On-line version, Jan 2013.
[24] Rajkumar, C. Lee, J.P. Lehoczky, D.P. Siewiorek, "Practical Solutions for QoS-Based Resource Allocation". In Proc. of 19th IEEE Real-Time Systems Symposium (RTSS 98). Madrid, December 1998.
[25] M. García Valls, et al. "A component model for homogeneous implementation of service-based distributed real-time applications". In Proc. of 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE - DANCE Workshop), pp. 267-272. Tozeur, Tunisia. May-June 2010.
[26] M. García Valls et al. "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems". IEEE Transactions on Industrial Informatics, vol. 9(1), pp. 228-236. February 2013.
[27] M. García-Valls, R. Fernández-Castro, I. Estevez-Ayres, P. Basanta-Val, I. Rodriguez-Lopez. "A Bounded-time Service Composition Algorithm for Distributed Real-time Systems". In Proc. of IEEE 14th International Conference on High Performance Computing and Communication - IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), vol., no., pp.1413,1420, 25-27 June 2012.
[28] M. García-Valls and P. Basanta-Val. "A practical solution for functional reconfiguration of real-time service based applications through partial schedulability". In Proc. of Int'l Workshop on Real- Time and Distributed Computing in Emerging Applications, REACTION 2012. Co-located with IEEE RTSS. Puerto Rico, USA. December 2012.
[29] M. García-Valls, P. Basanta-Val, P., I. Estévez-Ayres. "Real-time reconfiguration in Multimedia Embedded Systems". IEEE Transactions on Consumer Electronics, Vol. 57, No. 3, pp. 1280-1287. August 2011.

[30] Y. Benet et al., "RFC 2998. Integrated Services Operation over Diffserv Networks". IETF. http://tools.ietf.org/html/rfc2998/ November 2000.
[31] S. Shenker et al. "RFC 2212. Specification of Guaranteed Quality of Service". IETF. http://datatracker.ietf.org/doc/rfc2212/ September 1997.
[32] D. Cokuslu, A. Hermeurlain, and K. Erciyes. "Resource allocation for query processing in grid systems: a survey". In International Journal of Computer Systems Science and Engineering, vol. 27 (4). July 2012.
[33] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, “Developing applications using model-driven design environments”, Computer, Vol. 39, Issue: 2, pp: 33–40, Feb. 2006.
[34] A. Alonso, M. García-Valls, and J. A. de la Puente. "Assessment of Timing Properties of Family Products." Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families. Springer-Verlag, 1998.
[35] I. R. Quadri, A. Gamatié, P. Boulet, S. Meftali, J. Dekeyse. "Expressing embedded systems configurations at high abstraction levels with UML MARTE profile: advantages, limitations, and alternatives". Journal of Systems Architecture, vol. 58(5), pp. 178-194. April 2012.
[36] T. Arpinen, E. Salminen, T. D. Hamalainen, M. Hannikainen. "MARTE profile extension for modeling dynamic power management of embedded systems", vol. 58 (5), pp. 209-219. April 2012.