More Robustness and Flexibility for FPGA Based Networked Embedded Systems through Hardware Indirect Proxies

J. Barba, F. Rincón, J. D. Dondo, D. Fuente, F. Moya and J. C. López
University of Castilla-La Mancha, School of Computer Science, Ciudad Real (Spain)
Email: jesus.barba@uclm.es

***Abstract*—An indirect proxy is the name of a special kind of component wrapper that is defined within the framework of a hybrid (hardware and software) object-oriented middleware for Systems-on-Chips (SoCs). Which makes indirect proxies so attractive for networked embedded designs is that they do not need to know the physical address of the destination in advance. Regardless the actual place the required functionality is deployed at (in-chip or somewhere in the network), the proposed infrastructure enables transparent communication from the communication parties perspective. This is especially suitable in changing scenarios such as those using dynamic reconfiguration or adaptive systems. Also, there are applications in the field of QoS in communications and transient or permanent error management for SoCs that get benefit from this approach. In this article, we develop the indirect proxy concept and explore its capabilities oriented to improve flexibility, dependability and performance of FPGA based networked systems. The feasibility of our proposal is demonstrated by the implementation of a prototype using the Xilinx technology.**

I. Introduction

More and more, the development of a state-of-the-art System-On-a-Chip (SoC) is becoming a huge task involving tens of software and hardware engineers. Because this reason, silicon companies tend to maximize the productivity of their teams at the same time they try to reuse as much as possible previous designs. With the arrival of the reconfigurable logic technology, new opportunities are available to designers, oriented to reutilization since dynamic reconfiguration allows changing the hardware structures during run-time.

There are lots of application fields that can make the most of the flexibility and extra computation capabilities provided by FPGA (Field Programmable Gate Array) devices. Real-time/industrial control , image and video processing scientific/high performance computing or adaptive systems are examples of such FPGA ecosystems. Lately, FPGAs technology have also started to play a prominent role in networked embedded systems. Reconfigurable technology is being applied to the new challenges that networked embedded systems introduce, such as security remote/run-time management deployment, scalability or dependability, just to name a few of them.

Besides the improvement of processing performance in many ecosystems, including networked embedded systems, a reduction of the operation (i.e. energy consumption) and maintainability costs is observed due to the use of FPGA chips. These attractive features have promoted the appearance of hybrid platforms, for embedded distributed applications, that tightly couple a small processor with a reconfigurable fabric. Nowadays there is a variety of solutions based in this approach both commercial or ad-hoc [1, 2]. However, the functionality and communication infrastructure has to be tailored to fit with the current application requirements and it is usual that sophisticated ad-hoc control mechanism needs to be developed in order to obtain the best of such dynamic platforms [3].

978-1-4799-5743-9/14/$31.00

In this paper, we leverage the concept of indirect communication to ease the development of adaptive systems using reconfigurable technology at the network nodes. To accomplish this goal, two main concerns must be addressed:

1) The seamless integration of reconfigurable hardware functionality into the embedded distributed system. To cope with this problem we rely on our previous work on a Hw/Sw middleware for SoCs; the Object Oriented Communication Engine (OOCE) [4].
2) The implementation of a cross domain location service that spans both in-chip and embedded network domains. To this end, a special Intellectual Property (IP) wrapper called indirect proxy for hardware components is implemented and a distributed directory as well.

With (1), application developers are provided with access and location transparency whereas (2) adds extra location independency from the actual deployment of the system modules across the network.

In our opinion, some of the difficulties on networked dynamic systems management stem from the fact that key physical parameters are not efficiently decoupled from the application layers. For example, those that concern changes on the location or state of each component in the system since any variation on this properties has an effect on the rest of modules that are logically attached with [5].

The proposed mechanism and associated architecture is comprehensive since it applies to both individual reconfigurable nodes and the complete embedded network.

Moreover, the indirect communication approach can be also applied to problems that are not only exclusive of adaptive systems: fault tolerance, replicated hardware management and robustness [6] or quality of service are analyzed in this work.

Finally, it is worth mentioning that the use of indirect communication can provide some degree of flexibility to

systems with no reconfiguration hardware support.

II. The OOCE middleware

OOCE is a middleware specifically designed for SoCs which is composed by hardware and software functional units that make communication between the components of the system transparent and homogeneous. OOCE is based on a distributed object model so that the actors in the communication process are seen as objects; entities that encapsulate their behaviour by means of methods.

The main components of a SoC platform compliant with the OOCE guidelines are:

- The General Purpose Processor (GPP) executes the system functionality implemented as software objects and a thin broker to enable bidirectional HW $\leftrightarrow$ SW communication.
- The FPGA fabric holds: (a) the components that implement the hardware objects; and (b) two transparent bridges that manage the traffic between the FPGA and the GPP and the rest of the network.
- The Remote Object Adapter (ROA) is responsible for offering integration with external objects acting as a transparent router and protocol adaptor.

The communication is standardized by OOCE since it is defined the format of the invocation messages that must be exchanged between the client and server objects. A unique, technology-independent Remote Method Invocation (RMI) protocol is established for any kind of object and a platform-specific mapping of the RMI rules is carried out by an interface compiler. Thus, the generation of the necessary adapters (proxies and skeletons) is performed starting from a functional description of the object.

Proxies and skeletons isolate clients and servers respectively from the particularities of the communication channel and invocation protocol. OOCE defines several hardware and software templates of proxies and skeletons for the supported communication semantics (synchronous, asynchronous, etc.). Therefore, to trigger an operation at any object in the system, the client only has to interact with its corresponding proxy regardless the nature (Hw or Sw) of the component implementing the required functionality (server).

In-chip object functionality can be easily accessible by remote ICE clients. ICE (*Internet Communication Engine*[1]) is a CORBA-like middleware for distributed networked computer systems. OOCE encoding rules for RMI messages have been designed to be compatible with a subset of the data-type system of ICE. Hence, external software ICE objects running on a PC or software/hardware OOCE objects from an outer OOCE SoC can invoke internal objects. Invocations from inside the chip to an outer server object are also possible.

III. Indirect proxies

As introduced in the previous section, proxies play a key role in the communication process between objects. A proxy mirrors the server interface but in the client's side, providing the object with the illusion of interacting directly with the server. In OOCE, there are software and hardware proxies which offer an interface to initiate a method invocation: function call in software and signal activation in hardware. The proxy builds the invocation message according to the OOCE specifications.

The header contains the addressing information, basically an *Object Identification Number* (OId) that uniquely selects the destination of the invocation within the current OOCE SoC. Other header fields are: (a) the operation required (OP); (b) a request identification number (rqID) for out-of-order response management, if necessary; (c) whether the response is the result of a previous request; and (d) the OId of the client, which is needed to deliver the response correctly.

The body of the message codifies the method parameters (or returning values in the case of a response) and the context (optional) according to the OOCE data encoding rules. The context is a dictionary, that is, a sequence of key-value pairs that can be used to embed configuration or QoS parameters in the message.

The header is used to build a bus address which is used in hardware invocations (within the FPGA fabric). Such address is recognized by the skeleton of the hardware object in charge of processing the message.

There are two types of proxies; *direct* and *indirect proxies*. Direct proxies knows the destination OId and, thus, they have all the required information to build an invocation message at run-time. This type of proxies are used when the relations between the system objects are known at design time.

However, in direct proxies the target OId is fixed and cannot be changed at run-time which makes them unfeasible in dynamic scenarios. On the contrary, *indirect proxies* do not maintain any information regarding how to find the server (its OId). So that, in order to obtain a valid reference of the component, which implements the functionality to be invoked, an indirect proxy makes it use of a *location service* which provides the client object with the remote reference. In section IV, several applications of indirect proxies are described, starting with the management of the reconfiguration process in FPGAs. Indirect proxies facilitate the development of services for adaptive systems, based on the reconfigurable technology, decoupling the communication process from changes in the system configuration.

Next, the indirect communication process is introduced before giving a detailed explanation of the location service.

A. The indirect communication process

The indirect communication process may be compared to the *Domain Name Service* that governs the translation of domains to Internet Protocol addresses in Internet. Before connecting to a certain server (i.e. example.domain.com), the provided name is resolved to an address at the transport level which is used to connect with the remote endpoint.

In OOCE, each logical object in the application model is given a *Global Object Identifier* (GOId) which is composed by the *Object Identity* (the class of the object) and a *Global*

[1] http://www.zeroc.com

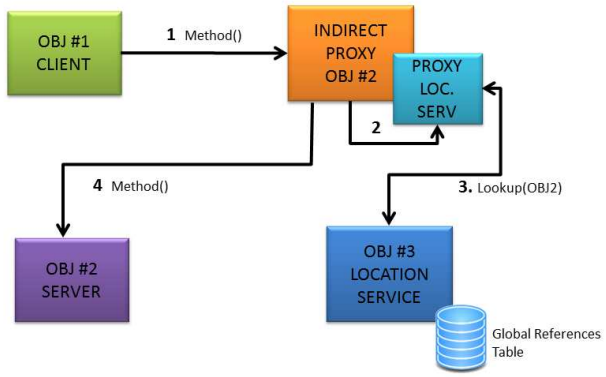

Fig. 1. Indirect communication: translation of object logical references to physical identifiers.

Object Identification Number (GOIN) which is unique for that type of object for all the active OOCE SoCs in the networked embedded systems. Recall that whereas the LOId is used as part of the physical address to reach the object within an OOCE SoC, the GOId is a logical identifier that selects one object instance of a certain in the whole system.

Indirect proxies only know the GOId of the server object it has to contact with, so the GOId must be first translated to a valid OId just as the domain names are first translated to Internet Protocol addresses.

The complete process can be summarized in the following steps:

- Step 1. The client module initiates the invocation on the indirect proxy as if it were interacting with the actual server module.
- Step 2. The indirect proxy checks whether it has the server's OId cached from a previous invocation. If it is present, go to step 4. If a valid OId is not available, go to step 3. An invalid or nonexistent OId scenario might happen due to: (a) a first-time invocation or; (b) a previous error trying to reach the server object.
- Step 3. The indirect proxy queries the location service for a valid OId given the GOId. On success, go to step 4. In case of failure, the error is communicated to the calling object.
- Step 4. The invocation process continues. If an error happens (server core does not respond or returns a failure code, etc.) the indirect proxy marks the cached OId as invalid, then go to step 3.

Fig. 1 depicts the actors involved in the indirect communication process and the main steps for an error-free indirect invocation. In this scenario, object #1 needs to invoke the *Method* on object #2.

It is worth noticing that the OId provided by the location service once finished the translation process might correspond to any of the types of OOCE objects (hardware or software) and it could be deployed in the same OOCE SoC or remotely somewhere in the network. OOCE communication services do the rest of the work and object #1 has not to be aware of these details.

TABLE I. DEFINITION OF THE GLOBAL REFERENCES TABLE FIELDS.

Fields	*Meaning*
GOIN	The Global Object Identification Number
Identity	The type of the component (the object class)
OId	Part of the bus/physical address
TimeStamp	The last time a reference of this object was given
NClients	A counter, the total number of references given

Although indirect communication introduces a delay (steps 2 and 3 in Fig. 1) compared with the direct method invocation, it introduces an extra degree of freedom avoiding the use of physical parameters associated with a specific configuration of the system. From the point of view of adaptive systems using FPGAs in a networked environment, the contribution of the approach is key since developers only have to focus on the design and implementation of the object functionality.

B. The location service

The location service is the OOCE subsystem in charge of managing the dynamic delivery of the server OIds. To this end, the location service maintains the *Global References Table* (GRT) which contains the necessary information to provide the best remote reference according to the established design criteria. Table I shows the GRT fields that relates GOIDs (GOINs + Object Identity) with their corresponding OIDs.

It is important to notice that it may exist more than one table entry pointing to the same physical base address or OIDs. This is true when the functionality of two or more system objects (logical objects) is assumed by the same physical object (hardware or software). This configuration might be appropriate, for example, to reduce the complexity of the FPGA design by concentrating functionality on a few IPs, although the clients only see the logical view, as a collection of individual objects.

Any OOCE object in the system is able to perform a query on to the GRT in order to find a reference to another object. This task must be performed through the search methods implemented by the location service.

The *lookup* method returns the OId assigned to the GOId provided as the input parameter. This is a one-to-one relation in the GRT.

The GRT field *Identity* is also used to resolve remote server references. The identity of an object is the base class from which it derives from, the type of the component. This parameter is not unique for each object since two or more system objects may belong to the same class. Thus, there might be a one-to-many relation when using this parameter to search an object reference.

Since there may be several instances of the same object type, an assignment policy must be defined. To reach this goal, the *TimeStamp* and *NClients* fields are necessary. OOCE location service offers three different assignment policies based on the treatment of the information of these fields: *NoPolicy* (first found, linear search), *RoundRobin* (the one with the oldest Timestamp) and *LessLoaded* (the one with the lowest NClient).

The location service is key to achieve our goal of a networked embedded platform which would be able of managing

the change in a natural, seamless way. Since the location service is defined as a regular object, there is no special consideration on the implementation of its functionality. This means the location service is just like another OOCE object in the system. Access and location transparencies provided by the middleware allows the location service to be implemented: (a) as a software object, local to the OOCE SoC; (b) as a hardware component, local to the OOCE SoC; (c) as a software object running on a PC (ICE server object) or external OOCE SoC and; (d) as a hardware component in an external OOCE SoC.

In this proposal, a full hardware implementation of the location service interface has been developed, providing the best performance results. Our prototype allows the federation of various location service objects enabling a decentralized query system of the references table.

Now, it will be shown how to improve the robustness, flexibility and communications in networked embedded systems design by means of using of OOCE indirect proxies and the location service.

IV. Applications of indirect communication

Indirect proxies introduce, unavoidably, an initial extra cost in the communication since there is necessary an intermediate step to get the reference to the destination. However, as it will be demonstrated later, this cost can be negligible with an optimal implementation of the location service. But the potential of indirect communication is more related with the transparency offered by the location service to: (1) manage multiple instances of the same functionality or objects; and (2) offer adaptability without stopping or replacing the current system. Five application scenarios are going to be exposed next. Without the support of the OOCE's indirect proxies and other facilities, the implementation of solutions for such challenges can become tedious and non-reusable.

A. Run-time failure management

One of the implementation strategies to cope with transient or permanent errors in distributed systems is the incorporation of core replicas for such critical functionality. Besides this, the reconfiguration logic arrays provide the system with the capability of self-healing in case of module failure. In both cases, it is usual that the control algorithm, which is in charge of monitoring and detecting errors in system modules, must be implemented (in software) by the designer. This fact forces the designer to spend divert his effort to the management of errors instead of concentrating on the resolution of the problem.

Indirect proxies offer to the designers a mechanism to hide the system failures, letting users and applications to complete their tasks even though some parts of the system are down. Once an indirect proxy detects that something goes wrong with the server, it invalidates the GRT entry at the location service and subsequent queries claiming for a new instance of an object of the same class are redirected. As it has been seen before, the GRT might has more than one entry of a certain type of object (*Identity* field) and the *lookup_type* method will provide always a valid reference.

The client object will never be aware about a change of the server instance used, the process remains transparent to it behind the operation of the indirect proxy. Only in the case that a new valid reference could not be returned, the error is notified back to the client.

OOCE indirect proxies are able to detect malfunction of server module using an internal timer that raises an exception when the expected response from the servers does not arrive (i.e. due to a failure in the object implementation or the routers or bridges that must drive the message to its destination). The middleware can also generate system errors to notify clients of temporal error conditions on servers such as *server overloaded* or *server does not accept incoming requests*. There are special protocol messages that embody the exception and are handled by the OOCE proxies.

B. Quality of Service

A way to improve system performance is the use of the location service to better balance the workload of servers. As it has been explained in the previous section, the Global References Table stores statistics of the number of references returned and when. The field *NClients* is a very simple metric, it assumes that the more the number of potential clients of a server the more is its workload.

Of course this is a rule that might not obey the behaviour of some applications, but the design criteria followed in this initial version of the location service is to obtain an optimal hardware implementation (minimum resources and minimum power consumption). Future versions of the location service will gather information concerning the actual load of the servers in real time. The middleware skeletons functionality are being extended to periodically communicate their status to the rest of the middleware core infrastructure and services. However, the application of this new feature should be carefully evaluated since the network and local buses could be flooded easily.

On demand server instantiation provides flexible computation capabilities to satisfy a peak of workload. The combination of indirect communication with run-time hardware reconfiguration (see later section IV-E) is an elegant response to the design of the infrastructure to support such service. As mentioned in the previous subsection, the OOCE middleware skeletons raise an exception of error condition message when they are about to reject new invocation petitions. Those message are system error messages that are delivered to all the core middleware components. When a condition of this nature is detected, a new instance of the server object is deployed with the help of the *Dynamic Reconfiguration Service*. Then, the new reference to the recently created instance is added in the GRT, making it accessible to the rest of the objects.

C. Migration of functionality

One way or another, all the depicted scenarios presented in this section are related with changes in the location of components. With *migration* we identify those challenges regarding *movement* of functionality. Self-adaptive systems are constantly changing in order to achieve the optimum status that allows to reach their goals: minimize the consumption of energy, maximize security, etc.

For example, the control of a distributed application decides to move a component from one network node to another

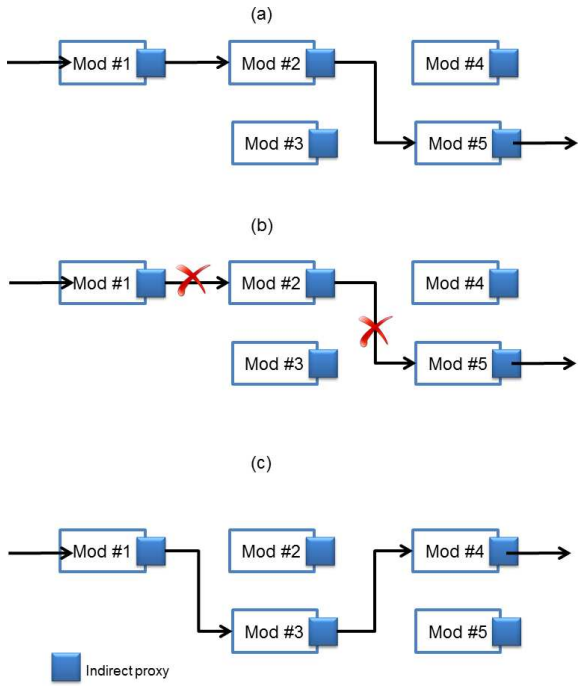

Fig. 2. System reconfiguration using indirect proxies

one closer to the principal client of such server. The reason can be varied: for example reduction of latency in communications or optimization of the network bandwidth. Mobility can also take place in-chip. Let us assume an object with two implementation alternatives; hardware and software. The application might decide to use both alternating between hardware and software attending, for example, to power consumption criteria or another QoS parameter.

So, how would these be done without the support of indirect communication? The intervention of the network administrator or maintenance operations of the systems would be required. With the presence of indirect proxies in hardware, transparency of migration is provided since the indirect wrappers are able to update the physical reference to the server on their own.

After moving the functionality from a location in the chip or network to another one, the any GRT entry in the system pointing to the old server reference is invalidated. The indirect proxy will detect an error the next time is going to invoke the server, forcing a new query against the location service to get the new reference. The new value of the OId is update by the actor responsible of the migration process and the instantiation of the new instance. For example, the control of the Dynamic Reconfiguration Service for hardware components or the Software Object Adapter.

D. Flexibility and run-time static adaptation

A different strategy to improve system flexibility or adapt its functionality without changing the number of active object instances is called *run-time static adaptation*. A static object is an object which never is going to be evicted. Nevertheless, using indirect communication, the references to such object can be modified at run-time and it can be temporarily inaccessible from other objects.

Now, let us think of an application that performs some preprocessing on the data acquired by the sensor nodes in a Wireless Sensor Networks before reaching the control. The logical view of the preprocessing chain can be similar to the one represented in Fig. 2.a with several objects of components, each implementing a single step. Again, each of those modules can be implemented in hardware or software, deployed in a single network node or various.

Only tweaking the values of the OId references at the GRT and invalidating the cached OIds at the indirect proxies (Fig. 2.b), the configuration of the processing chain can be easily altered (Fig. 2.c). The only requirement is that the replacement for the module must implement the same interface, so that the compatibility is assured.

Besides these types of applications where the adaptation is network-wide, in-chip behaviour might also be varied making it use of the same principles (i.e. encoder/decoder).

E. Dynamic reconfiguration support

Many of the scenarios described above require changes in the location of the component functionality to accomplish the system goals. Thus, additional support to create new instances of OOCE objects is needed, making the most of the reconfigurable logic technology. The efficient management of the reconfiguration process of the FPGAs is the cornerstone of adaptive and dynamic systems.

In [7] the reader can find the details of a novel architecture, development workow, and modelling approach for dynamically recongurable systems management. This solution is built upon the OOCE system-level middleware and offers a global solution to the partial reconfiguration process.

The *Dynamic Reconfiguration Service* is implemented in each FPGA node and is composed by the *Reconguration Controller* (RController) and the *Factory*. The RController is designed to manage the reconguration process within the system and offers a set of services to the users, such as reconguration, persistency and object-location services. Here, the indirect communication approach and indirect proxies are fundamental.

V. Validation of the Proposal

To demonstrate the viability and efficiency of the proposal for embedded networked systems, a prototype of the distributed platform for adaptive applications has been developed. The network nodes are built on a XUPV5-LX110T board which has a Virtex-5 XC5VLX110T FPGA chip with 110K logic cells that supports partial dynamic reconfiguration. This platform is intended only for prototyping and design exploration purposes of the solutions developed in this work.

The prototype consists of four XUPV5 nodes plus one PC which runs a control software that communicates with the network nodes. Using the PC application, bitstream deployment, GRT management of the different location service instances, OOCE SoC object invocation and other remote control operations can be performed. There is a GUI or control panel to help with these tasks.

Each FPGA node has a complete OOCE middleware infrastructure that comprises: (a) software layer to run objects

TABLE II. SYNTHESIS RESULTS.

Component	*LUTs*	*FFs*	*Slices*	*Max. Freq.*
Location service (synchronous)	134	57	102	135
Location service (asynchronous)	206	66	157	100
Indirect Proxy (synchronous)	8	24	10	225
Indirect Proxy (asynchronous)	72	41	36	200

on a MicroBlaze processor; (b) a Remote Object Adapter for external communications and; (c) an instance of the Dynamic Reconfiguration Service. On system startup there are no instances of OOCE objects at the application level, and they have to be added using the control panel from the PC and the local Dynamic Reconfiguration Services.

Three classes of objects can be instantiated (the bitstreams and binaries are stored at the control node). The behaviour of these objects is very simple; each one invoke periodically a *dummy* method on the other two. The elapsed time between invocations can be configured. Of course, communication is performed using indirect communication so each object instance has two indirect proxies and a skeleton associated with it. The ultimate goal is to create communication patterns between object instances within a node or inter-nodes.

Error injection is another functionality of the control software. Objects can be disabled to simulate a server breakdown, OId references invalidated, etc.

This tool allows the emulation of scenarios directly on the networked embedded platform. Currently, a new version of the control panel is being developed so it can be supported the deployment of any user functionality and adding monitoring features to gather actual statistics directly from the real application.

Regarding the overhead in term of resources that implies the use of the core infrastructure, synthesis numbers say that all the core components represent less that 1% of the total logic available on the FPGA node. The logic of a single indirect proxy is negligible, just a few tens compared with the more than 17K slices of the XC5VLX110T FPGA chip. Table II summarizes the results obtained with the ISE Design Suit 13.4 from Xilinx. Both synchronous and asynchronous versions of the indirect communication components have been developed. Asynchronous invocations needs of extra resources because of the extra memory used to queue invocations in the skeleton or store responses in the proxy due to out-of-order arrival of the return values.

The location service component is fundamentally formed by two CAMs and a RAM. The OId values and data concerning object statistics are stored into the RAM. The RAM is indexed by the output of both CAMs. The first CAM maintains the GOIds of the currently active objects, the second one maintains the Identities of such objects and its output is able to notify multiple-matching. Special logic implementing the reference assignment policy analyzes the data of each memory hit and takes a decision.

VI. CONCLUSIONS

In this paper, we have dissected the applicability of a new kind of wrappers for Intellectual Property hardware components in networked embedded SoCs. Indirect proxies handle communication with other components independently of their actual deployment in the network which makes them suitable for dynamically changing scenarios. There are many applications that can benefit of this approach, but we have focused in this work on those features specially attractive to adaptive and distributed systems: partial reconfiguration management, transparent off-chip communication, etc.

In summary, indirect proxies allow to: (1) move the functionality of the server cores with freedom; (2) implement simple and efficient mechanism to manage core replication and transient errors; (3) define mechanisms to improve communication throughput; and (4) provide certain degree of flexibility to those systems that originally were not designed for.

This is possible without a special effort of the designer, programmer or user. They benefit from the transparency provided by the location service, the indirect proxies and the OOCE facilities that easy the design, programming and runtime adaptation of the system platform.

ACKNOWLEDGMENT

This research was supported by the Spanish Ministry of Science and Innovation under the project DREAMS (TEC2011-28666-C04-03).

REFERENCES

[1] Y. Krasteva, J. Portilla, E. de la Torre, and T. Riesgo, "Embedded runtime reconfigurable nodes for wireless sensor networks applications," *Sensors Journal, IEEE*, vol. 11, no. 9, pp. 1800–1810, 2011.

[2] H. Hinkelmann, P. Zipf, and M. Glesner, "Design concepts for a dynamically reconfigurable wireless sensor node," in *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, 2006, pp. 436–441.

[3] M. Ullmann and J. Becker, "Communication concept for adaptive intelligent run-time systems supporting distributed reconfigurable embedded systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, pp. 8 pp.–.

[4] J. Barba, F. Rincón, F. Moya, J. D. Dondo, and J. C. López, "A comprehensive integration infrastructure for embedded system design," *Microprocessors and Microsystems*, vol. 36, no. 5, pp. 383 – 392, 2012, special Issue on Design of Circuits and Integrated Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S014193311200024

[5] D. Gohringer, M. Hubner, V. Schatz, and J. Becker, "Runtime adaptive multi-processor system-on-chip: Rampsoc," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1–7.

[6] K. Waldschmidt and M. Damm, "Robustness in soc design," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. 27–36.

[7] J. D. Dondo, J. Barba, F. Rincón, F. Moya, and J. C. López, "Dynamic objects: Supporting fast and easy run-time reconfiguration in fpgas," *Journal of Systems Architecture*, vol. 59, no. 1, pp. 1 – 15, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138376211200086