Implementation of context-aware network architecture for smart objects based on functional composition[1][2]

Jose Luis Ferrer, Xavier Sanchez-Loro, Anna Calveras and Josep Paradells

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB)
JordiGirona 1-3, 08034 Barcelona, Spain
i2CAT Foundation, Gran Capità 2-4 (Nexus Building), 08034 Barcelona, Spain

{jlferrer,xsanchez,acalveras,josep.paradells}@entel.upc.edu

Abstract. Lack of flexibility of current Internet architecture led researchers to come up with new paradigms for a novel Internet architecture, which would be able to reduce complexity and increase flexibility compared to current Internet architecture. Functional composition is a promising approach to flexible and evolvable architecture design. The idea is composing complex protocol suites by dynamically bind and arrange different functions to obtain certain behavior. Herein, we present the implementation of a context-aware network architecture based on functional composition for smart objects. A sub-set of those basic functional blocks has been implemented and validated on an experimental testbed using different network topologies.

Keywords: Future Internet, functional blocks, context-awareness, smart objects, constrained devices, semantic routing.

1 Introduction

Nowadays, several different proposals of Future Internet (FI) architectures exist in Europe and worldwide that are taking the "clean-slate" path, redefining the architectonical principles of Future Networks (FNs) from scratch. Hence, lack of flexibility of current Internet architecture leaded researchers to come up with new paradigms for a novel Internet architecture, which would be able to reduce complexity and increase flexibility compared to current Internet architecture. But, at this moment, it is not clear which of these architectures neither which architectural paradigm will succeed or prevail as the new Internet architecture.

Functional composition is a promising approach to flexible and evolvable architecture design [1][2]. The idea is composing complex protocol suites by dynamically bind and arrange different functions to obtain certain behavior. The key aspect here is granularity and scope of functions. On one hand, the idea of

[1] This work is supported by the Spanish Government through the MICINN and FEDER project TEC2009-11453.

[2] The authors are grateful to the reviewers' valuable comments that improved the current paper.

modularizing the stack is catching the great attention in the research community. The idea is to extract the functionalities out of the current network stack and place them as needed, by using this approach it will ease the introduction, reuse, change and placement of functionalities (e.g. application level, network level, etc.) and flexibility in terms of inclusion and exclusion of function with respect to application demands. Another similar idea that is catching up is using a flexible model of layers with dynamic bindings, opposed to the static binding between stacked modules/layers being used today. The idea here is to optimize the architecture for flexibility, designing it to support any protocol stack by using inheritance schemes (class-based), polymorphism and dynamic binding between modules/layers. This approach also aims to make a flexible functional composition, but with coarse-grained granularity of layers instead of small networking functions.

The paper is organized as follows: in section 2 we discuss the possibilities of functional composition to implement FI architectures suited for smart objects. Section 3 gives a brief overview design of our architecture proposal (see [3] for more details on the design of the architecture). Section 4 details the implementation of the architecture for constrained devices.

2 Functional Composition in Smart Objects

Functional composition architectures for FI have the potential to be able to cover the whole networking spectrum in its wider sense if they are provided with extensible, adaptable, customizable and personalized roles, vocabularies and control mechanisms. They have the potential to converge the different “Internet of” visions into a single architectonical paradigm; thus paving the way for convergence of heterogeneous networks and integration of smart objects and other constrained devices in the general framework of the Internet.

Hence, solutions should be designed and standardized with extensibility, flexible and loose-coupling as main design premises. Therefore, software-derived mechanisms for data abstraction, modularity, inheritance/delegation, dynamic bindings (opposed to static ones) and polymorphism should be applied in order to allow the evolution and change of the different aspects of the solution (roles, vocabularies and control mechanisms). By providing tools for changing the dynamics and behavior of networks and protocols FI architectures will have realistic means to evolve without requiring major breakthroughs, costly transition plans or architectural reformulations.

In accordance with this idea, we should design solutions for worst-case scenarios, such as constrained environments with capacity and power restrictions and hostile mediums (e.g. smart objects, actuators, sensors), and then extend them to work in more favorable environments without capacity restrictions. Therefore, if we can design flexible architectures that are adaptable to context variations and that work with minimalistic devices in constrained networks, it should be easier to extrapolate those architectures to work in more unconstrained environments, without capacity restrictions, such as wired computer networks. Furthermore, restricted devices like sensors/actuators and objects are becoming increasingly predominant in the Network

(and will continue to grow in numbers); we cannot therefore ignore their impact on networking applications and should design a model capable of accommodating them.

3 Architecture Overview

In [3] we propose a clean-slate Role-Based Architecture [4] that advocates breaking current protocols in its fundamental pieces: those individual functions commonly used in networking protocols (i.e. acknowledgments, sequence numbers, flow control, etc.).The idea is creating new protocols using these atomic functions or services as basic building blocks. This way, the proposal uses ensembles of basic services to provide advanced communication services. We define those basic building blocks as Atomic Services (AS).

This shift of paradigm requires extensive research on feasible techniques for building up optimized protocols from scratch according to different criteria, ranging from energy efficiency to context-awareness. This reasoning should take into consideration the characteristics of the surrounding context (device capabilities, available network and computation resources, location and environment, etc.) solving the following issues [5]: which behaviour and outcome is desired; which functions are most suitable to achieve expected outcome; where they should be allocated; which mechanism (protocol, language/ontology) will be used to allocate and configure functions along communication path.

Other issues approached by the architecture are: discovery of desired services according to their semantic description; mechanisms for context interchange; resource reservation and ontologies for describing context characteristics (node, link, service, etc.), QoS agreements, service/resource description, locator/identifier schemes, etc. [6].The service discovery and AS allocation is performed by means of negotiation protocol based on reactive ad-hoc routing protocols suitable for smart objects which creates communications on demand.

For ontology and vocabulary support, we define a set of basic vocabularies for describing node capabilities, service and resource characteristics and QoS requirements. In order to support different environments and visions, this set of services/roles should be extended. Likewise, these vocabularies are designed with restricted devices in mind and, as such, are quite limited in their scope.

4 Implementation Design

The design of the architecture is divided in modules that run on event based operative system for constrained devices [3]. We have extended the architecture to be platform independent using an event based scheduler and a Medium Access Control (MAC) module able to inject and receive datagrams from different physical network interfaces. Furthermore, the architecture provides a framework to create functional blocks, i.e. ASs, and provide information about its characteristics to be utilized by allocation algorithms when establishing a new communication.

This section is divided as follows. First of all, we discuss the main patterns required to implement the functional blocks. Secondly, we detail the structures utilized to register and select those blocks in the architecture. Then, we describe the available interfaces to interchange information between the different functional blocks in an active communication. Finally, we describe a preliminary validation of the framework using several nodes with a basic set-up of functional blocks.

4.1Atomic Services Patterns

In addition on creating taxonomy of types of basic network functionalities for services (e.g., cyphering, error control, etc.), we have identified the following patterns as the basic operations that are usually executed when implementing any networking protocol. These patterns should be available when executing an AS in order to provide flexibility and extensibility:

- *Payload data modification*. Services like coding, transcoding or ciphering mostly modify the payload from application data included in the datagrams. Therefore, the architecture must provide interfaces to the AS in order to access to packet payload independently on other AS executions.
- *Overhead data manipulation*. Network protocols require interchange and access to additional control data included as overhead in packets. Hence, the architecture must provide standard interfaces to access to packet metadata information.
- *Execution of periodic tasks*. Timers are required in order to evaluate certain conditions that could trigger events like retransmissions or QoS computations. Hence, the AS executions must be able to create timers to trigger those events.
- *Generation and processing of control data*. In addition to overhead control information added to data payloads, some services must be able to generate control data interchange, like the negotiating phase of security protocols.

4.2. Atomic Services

Atomic Services (AS) constitute the functional modules of the architecture utilized to set up communication paths. Once a communication has been established, the allocated AS are executed sequentially. We define the procedure of creating this list is as AS linkage (see Fig. 1). For each path created, each node maintains a linked list of the AS to be executed locally. Each network function is created using two AS blocks that represent the two endpoints of the network functionality. Each AS block can be configured as input, if processes receiving packets; output, if processes outgoing packets; or both in case that the AS is executed only in an intermediate node (e.g., a transcoding functionality).

We have classified AS in different groups or types, according to its main functionality. Examples of AS groups are error detection, transcoding, retransmission or acknowledgement [6]. For each of the AS groups, there exist different implementations for the same functionality, defined in the architecture as AS configurations. For instance, Stop-and-wait Automatic Repeat Request (ARQ) and Go-back-N ARQ are two different configurations of the retransmission functionality.

Therefore, each AS block is uniquely identified by its type and configuration. Currently, each AS type and configurations in the architecture are identified using 1 byte, enough to run a basic set on constrained devices but can be extended to allow a greater set of implementations. As a result, organizations and developers can provide a wide range of AS, where basic and simpler AS implementations can be applied in constrained devices and complex ones in carrier-grade networking equipment.

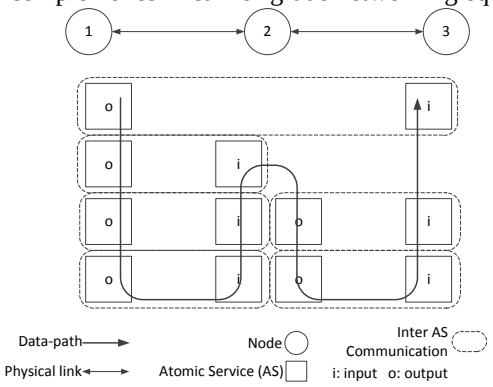

Fig.1. Example of 2-hop AS linkage.

4.2.1 AS Selection Rules

The selection of appropriate AS configurations become crucial to obtain the desired communication behavior based on applications requirements. Different approaches exist in the literature, authors in [7] propose and out-band signaling infrastructure to apply data-path services based on defined QoS policies and services conditions to be executed through nodes in the path. While in [8] is defined a framework to create offline functional composition of building blocks based on the weighted combination of multiple QoS attributes provided by each building block implementation.

We have defined the selection of the suitable AS configurations based on processing the context information available to nodes. In the case of smart object or infrastructure-less networks, context information is obtained via the negotiating protocol. That information is expressed by means of the context attributes, including node characteristics (available memory, battery level,...), link characteristics (bandwidth, delay, jitter, quality metrics, load,…) and network characteristics (e.g, domain). The selection rules consist on a combination of comparison operations using the values of context attributes as inputs.

The computation of the rules is performed as follows: first, each AS configuration that offers the QoS required for the application is evaluated using a cost function in order to determine its suitability to be allocated in a new communication path. Defining $AS_{i,j}$as the configuration *j* for AS of type *i*, the equation (1) defines the cost function, $C_{AS_{i,j}}$, computed for all the applicable $AS_{i,j}$ in a communications. This cost function is the sum of a set of *N* weighted comparison operations for a set of context attributes $(a_{i,j,k})$ which represent the logical function, denoted as $logicf\left(a_{i,j,k}, A_{i,j,k}, CR\right)$ with weight $\omega_{i,j,k}$, $A_{i,j,k}$ as the constant value to

compareand *CR* as the comparison relation to compute $(=, \neq, \geq, \leq, >$ or $<)$ (see equation (2)).

$$C_{AS_{i,j}} = \sum_{k=0}^{N-1} \omega_{i,j,k} * logicf\left(a_{i,j,k}, A_{i,j,k}, CR\right) \quad (\mathbf{1})$$

$$logicf\left(a_{i,j,k}, A_{i,j,k}, CR\right) = \begin{cases} 0 & ; if\ a_{i,j,k}(CR)A_{i,j,k} = FALSE \\ 1 & ; if\ a_{i,j,k}(CR)A_{i,j,k} = TRUE \end{cases} \quad (\mathbf{2})$$

Finally, an AS configuration $AS_{i,j}$ is applicable for a communication path if the $C_{AS_{i,j}}$ is greater than a defined threshold ($Th_{AS_{i,j}}$). Note that the selection of weight and threshold values requires a previous learning or experimental phase in order to tune the allocation rules. For instance, the configuration CRC-16 for the Error Detection (ED) functionality$AS_{ED,CRC-16}$ can be utilized in a wireless link if the context attribute SNR ($a_{ED,CRC-16,SNR}$) is lower than a specified value.

4.2.2AS Register (ASR)

The architecture provides a memory space where all the implementations of $AS_{i,j}$are registered using a linked list of structures that include the function callbacks executed by the. $AS_{i,j}$ during run-time. That list is loaded with all available AS configurations when the architecture is initiated. Each $AS_{i,j}$ implementation must include in the register the following callbacks functions:

- *as_init()*. Called when the AS is allocated in a node during the negotiation phase. It is utilized to initiate AS memory structures or communication attributes.
- *exec_in()*. Executed for an AS configured as input after receiving a new data packet.
- *exec_out()*. Called by AS which are configured as output, i.e. data is transmitted to the network.
- *exec_inout()*. This function callback is utilized by AS that act as input and output at the same time (i.e., executed in the same node).
- *exec_timer()*. When a timer has expired for a specific AS block, this callback is utilized to run periodic tasks.
- *exec_control()*. Control packets receptions are processed via this callback.
- *as_destroy()*. Executed when closing the communication to free any memory allocated at AS initiation.

After selecting the AS configurations to be allocated in the data path, the node originator of the communication requests distributes the configuration information of the AS to the rest of the nodes of the data-path. Then, each node creates the sequential list of callbacks for the selected AS blocks in the new allocated communication. Afterwards, the data is transmitted from source to destination nodes.

4.2.3 Data-Path Processing

Once established the data path for the communication, the AS linkage provides a list of AS callbacks to be executed in each of the nodes among the data path. The execution of the AS processes the communication data and control information

included in two types of packets: control and data packets, respectively(see Fig. 2).Each time a packet traverses a node, the AS Execution Manager in the responsible to execute sequentially the callbacks for each AS in the data path accordingly to its configuration and direction (input, output or both).Control packets include control data transmitted to an AS addressed using the 1 byte address, which is generated after AS allocation; while data packets include a header overhead with the following fields:

- *num* (1 byte). Indicates the number of parameters.
- *len* (1 byte). Total length of the overhead.
- *Type Value (TV) Parameters*. Parameters with fixed length (1, 2, 4 and 8 bytes) defined by its type, adding 1 extra byte of overhead indicating the parameter type.
- *Type Length Value (TLV) Parameters*. Parameters with variable length, adding 2 overhead bytes (TL).

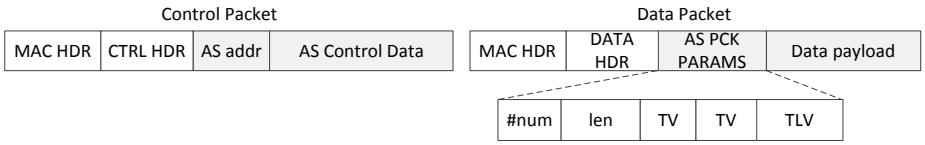

Fig. 2. Description of packets defined in the architecture with AS metadata.

The main interface utilized by the AS callbacks implementations is a common memory structure which is passed as argument to the callbacks implementations. This structure is allocated for each AS at AS linkage, requiring a total of46 bytes (using 32 bits architecture. The structure includes flags (4 bytes) and memory address pointers to structures that allow the AS to perform its patterns (10 pointers in total). The main structures required by the AS are the packet payloads, packet metadata (i.e., packet attributes), control data and global connection control data (i.e., connection attributes). The data path processing in a node takes the following stages (see Fig. 3):

- *Data packet reception at the MAC Layer Bundle*. A new packet is received from the MAC layer, encapsulating the data packet. The MAC bundle extracts the MAC Header and sends the packet to the Network Module of the architecture.
- *Processing of data packet at Network Module (NM)*. The NM looks for the registered communication in the node that matches the data header. On positive matching, it sends the data payload to the AS Execution Manager.
- *Execution of AS linkage in the node*. First of all, the de-serializer extracts the AS metadata included as overhead in the data packets. Secondly, the AS list is executed sequentially. After each execution, the AS Execution Manager checks whether it is new control data generated to be transmitted to the network. Moreover, during any AS execution, the AS callback can add a timer to execute future tasks. Finally, the serializer adds the metadata (i.e. packet attributes after AS list execution) in the data packet to be transmitted in the network.

After data packet processing, the AS block can mark the packets status to be deleted or queued (in case that future actions are required). If AS do not set packet status to queued, the AS execution Manager removes the packet from queue after it has been processed.

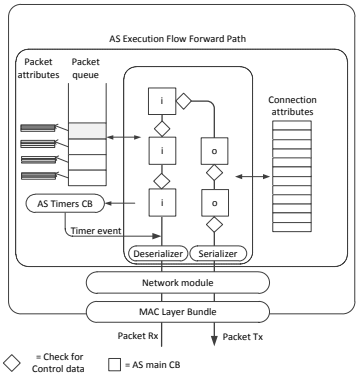

Fig. 3. Example of a communication data-path in a node.

4.2.4 AS Communications

There are two possible methods to interchange data between AS blocks in a communication: inter AS communication, when AS blocks are allocated in different nodes; and intra AS communications, among AS blocks executed in the same node.

4.2.4.1 Inter AS Communications

The AS Execution Manager checks whether new control data has been generated after each AS block execution. When required, the AS Execution Manager creates the control packet and sends it to the network. Control packets transmissions are bi-directional in the AS linkage and can only be processed with two AS that share the same AS address, i.e., AS blocks that implement the same functionality but configured with the inverse behavior (input instead of output and vice versa).Fig. 4 shows and example of how control data is traversed through nodes using the reverse path until it reaches its destination AS block.

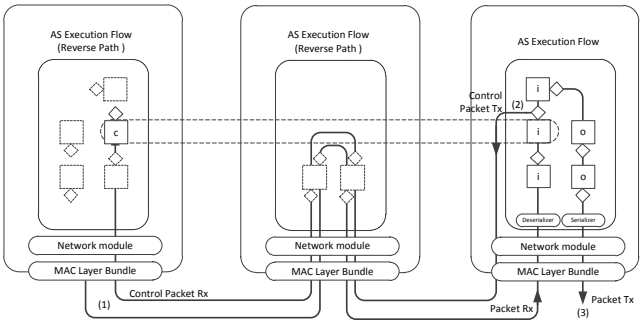

Fig. 4. Control packet processing through 2-hop communication.

4.2.4.2 Intra AS Communications

The intra AS communication is generated via a shared memory space of connection metadata data using two different structures: packet attributes and connection

attributes. Packet attributes include the metadata of current processed packet which is extracted by the de-serializer. After processing the AS linkage list, the serializer adds the packet attributes as data payload overhead (see Fig. 2). Connection attributes include global control attributes for each communication, creating a standard way to perform cross-layer operations between the different functions. Both packet and connection attributes schema can be extended to allow the architecture evolution.

4.3 Preliminary Validation of the Architecture

The architecture has been implemented in C language in Linux and OSX based distributions using an Ethernet MAC Layer Bundle with raw sockets and Berkeley Packet Filter (BPF), respectively. The interface network driver has been emulated in the MAC Layer Bundle running two separate threads to receive and send packets by the network module. The identifier for each node has been set to a fixed value and included in the MAC header of the MAC Layer bundle. We have implemented a total of 9 AS configurations grouped in 5 AS main functionalities as a basic sub-set in order to build reliable communications:

1. *Acknowledgement*: sequenced ACK, not-sequenced ACK.
2. *Data encoding*: base64 encoding.
3. *Error detection*: CRC-CCITT, CRC-16, CRC-32, RS-FEC.
4. *Retransmission*: Basic per packet retransmission.
5. *Sequencing*: Basic packet sequencing.

To validate the negotiation protocol, the AS Execution Manager and AS Selection rules, we have emulated different network topologies with up to 30 nodes by means of applying network address filters in the Network Module. The context information for each link has been emulated setting different link quality metric utilized to select the most suitable AS configuration. To be more precise, a link metric value of 1 indicates a very good link quality, while 0 indicates poor quality. For instance, we defined rules based on the link metric values to allocate the Acknowledgement and Retransmission AS configurations, resulting in allocation of AS blocks for these functionalities on the links where the link quality was below the 0.7 value.

A simplified version of the described architecture has been coded and compiled for the CC2430 platform [3], based on 8-bit architecture System on Chip (SoC). This exercise demonstrates by one side the feasibility of the approach and by the other the need of more powerful platforms to benefic of the capabilities of the platform.

5 Conclusions and Future Work

Functional composition paradigm stands as one of the most promising paradigm for FI architectures. Constrained devices like smart objects will be predominant in the future; accordingly, we consider that FI architectures should be supported in smart objects. In this paper, we have presented an implementation of context-aware functional composition architecture suitable to run on constrained devices. The

context semantic description (QoS, service, node and network) provides extensibility as it supports the easy adoption of future services provided by smart objects.

The Atomic Services (AS) are the basic building blocks of the architecture, each modular functionality is implemented as an AS configuration and registered in the architecture. Those AS are able to access dynamically to other AS data via the generic packet data headers and the global connection attributes, allowing a flexible cooperative and cross-layer access not supported in current architectures. A sub-set of those basic atomic services has been implemented and validated on experimental testbed with several machines and network topologies.

Currently, we are working in the implementation of additional AS configurations; in the extension of the AS interfaces and improving the previous 8-bit architecture to the MC1332x[3] platform (based on 32-bit ARM). Further research is required in order to analyze AS dependences and evaluate the performance of the AS configurations using different context attributes. Based on those attributes, we will design and implement different AS Linkage algorithms to be validated emulating different context attributes. Furthermore, final performance in testbed with constrained hardware platforms should be analyzed in order to validate the emulated network conditions. Finally, we plan to extend the architecture to be infrastructure based, using multi-interface elements that lead to a global connectivity.

References

1. Henke, C.,Siddiqui, A.,Khondoker, R.: Network functional composition: State of the art. In: Telecommunication Networks and Applications Conference (ATNAC). Australasian, pp.43--48 (2010)
2. Schinnenburg, M., Debus, F., Pabst, R.:Application of Functional Unit Networks to Next Generation Radio Networks. In Vehicular Technology Conference (VTC) Spring. IEEE 63rd. 7-10 May 2006, Volume: 1, pp. 147 -151 (2006)
3. Sanchez-Loro, X., Ferrer, J.L., Gomez, C.,Casademont, J., Paradells, J.: Can Future Internet be based on constrained networks design principles?. In: Computer Networks, 55, 4, pp 893—909. Elsevier North-Holland, Inc. New York, NY, USA(2011)
4. Braden, R., Faber, T., Handley, M.: From protocol stack to protocol heap: role-based architecture. In: SIGCOMM Compututer Communication. Rev. 33, 17–22 (2003)
5. Sanchez-Loro, X, Gonzalez, A.J., Martin-de-Pozuelo,R.: A Semantic Context-Aware Network Architecture. In: Future Network and Mobile Summit 2010. Florence, Italy (2010)
6. Sanchez-Loro, X., Ferrer, J.L, Casademont, J., Paradells, J., Vidal, A.: Proposal of a Clean Slate Network Architecture for Ubiquitous Services Provisioning. In: IEEE ICFIN 09. Beijing, China (2009)
7. Shanbhag, S., Wolf, T.: Automated composition of data-path functionality in the future internet. In: IEEE Network, 25, 6, pp.8—14.(2011)
8. Völker, L., Martin, D., Rohrberg, T., Backhaus, H., Baumung, P., Wippel, H., Zitterbart, M.: Design Process and Development Tools for Concurrent Future Networks. In: 3rd GI/ITG KuVS Workshop on The Future Internet, GI/ITG Kommunikation und VerteilteSysteme, Munich, Germany (2009)

[3]FreescaleMC13224V: MC1322x Platform in a Package (PiP). http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC13224V