A Formal Method for the Evaluation of Component-based Embedded Systems: Application to Technical Choices for CSTBox Toolkit

Daniel Cheung-Foo-Wo, Éric Pascual
Centre Scientifique et Technique du Bâtiment (CSTB)
Sophia Antipolis, France
daniel.cheung@cstb.fr, eric.pascual@cstb.fr

***Abstract*—When designing applications for sensor or actuator network management, deployed networks of equipments often use an underlying component-oriented approach, such as OSGi (formerly Open Services Gateway Initiative, now OSGi Alliance), which provides flexibility in terms of software development and maintenance. It also leads to combining sensors and actuators from different manufacturers using different protocols, libraries, and communication media. To make the interface between components compatible, additional software layers are implemented on top of heterogeneous libraries and protocols. Our intention in this paper is to confront our OSGi applied software solution coping with these heterogeneity and flexibility to contextual timing constraints in order to propose solutions in terms of OSGi extensions and generalization. We formally describe the performance function of the connectors of our toolkit called CSTBox (CSTB Sensing Telemonitoring Box) faced to soft real-time constraints derived from real-world deployments of European projects in domains such as ambient assisted living for the elderly and energy use in offices according to a simple classification of underlying middleware in terms of performance. Running applications, result analyzes, and timing measurements are performed on the SOC (System on Chip) card Raspberry Pi hardware for low-consumption and other embedded systems like routers and gateways.**

Keywords-component; soft real-time constraints; OSGi; D-Bus; sensor network data processing measurement;

I. Introduction

The problem addressed in this paper is derived from research projects [1]–[3] in ambient instrumentation of buildings. The objective is to provide software managing networks of equipments in the form of an OSGi-based toolkit for embedded systems in ambient assisted living and analysis of energy use. However, these domains impose some requirements on systems. Firstly, in order to protect *privacy* in home activity monitoring, external data transfer is securely kept minimal. Besides, the system must be robust to network failures to avoid information loss. As a result, most of the data processing is done in-place, which promotes the systems from simple data acquisition and transfer units to full processing units. Secondly, these domains concern monitoring for gathering statistical information and require sufficient *soft real-time* (RT) constraints on observation rate and reaction delay. Various applications from assisted living and energy use accordingly *share* a great part of functionalities, and emphasize the need for a component-based solution. The system must also be flexible in supporting multiple sensor networks connecting from local and remote sites. Furthermore, the hardware equipments, such as sensors, actuators and network coordinators, might differ because of their origin and their protocol. These *heterogeneous* equipments are thereby mixed for a given instrumentation of buildings (see Figure 1).

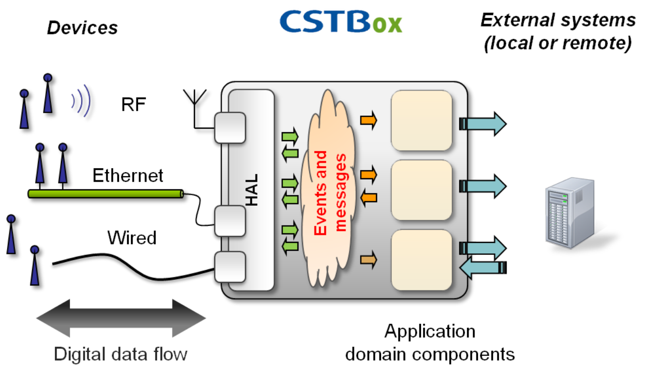

Figure 1. Operation principle of the CSTBox abstraction mechanism

On account of these requirements, our software toolkit is characterized by its assembly of soft RT components using heterogeneous equipments. We proposed a solution [4] tackling sensor-actuator heterogeneity by means of representing an *equipment* (decomposed into parts) as a set of readable or writable *state variables* associated with a *type*, an *identifier* of the associated part and the *state* of the equipment part.

This paper deals with the *confrontation of this OSGi approach to timing constraints induced by assisted-living and energy-use domains* and its consequences on the choice and the design of the component communication provided by the underlying OSGi platform, i.e., *software connectors*. The remainder of this paper is organized as follows: next section presents some related works. Section III introduces the model of our toolkit and presents a formal approach for scoring its performance. In Section IV, we detail the problem of applying constraints and the way it addresses the issue of the proper selection of connectors. We conclude in Section V on perspectives.

II. Related work

Software connector design and choice can be studied in several ways. For instance, connector choice can be achieved by querying an existing classification of connectors [5] organized in terms of the following six main dimensions: the amount of transferred data, delivery intervals, performance requirements, the number of consumers, access policies, and geographic distribution of components. A data distribution scenario is then expressed as a query against the above dimensions. The selection algorithm is thus tasked with deciding what available connectors will suit a given scenario, which is accomplished by maintaining a knowledge base of connector profiles. A connector profile contains metadata including locality, number of providers/subscribers, throughput, delivery semantics, and routing. The authors' focus is on automatizing the selection algorithm for fetching software connectors based on the key dimensions. Although automation represents a great benefit to software engineering, we are more interested in a formal tool helping the developer figuring out the adequate use of connectors.

Another related area of research relies on software architecture programming dealing with frequent updates. Reference [6] presents the design of connectors by combining *white-box* components which semantics and routing are known. The authors focus on mathematical properties, such as associativity of the combination, forming an algebraic way of designing connectors. They use this algebra to integrate, swap or remove functionalities expressed in terms of components. They consider that the set of all adaptations that may be deployed cannot be anticipated at design time, which is beyond our requirements. In such case, adaptations may interfere when they are combined. However, they model the assembly of components as a directed graph (data-flow) providing them with a formal tool to identify properties and errors. We consider designing a formal model compatible with this graph representation and algebra.

Finally, in a software engineering point of view, OSGi provides all the requirements expressed in the Section I. However, based on a qualitative experience, [7][8] decided not to use OSGi as a connector implementation for three reasons. Firstly, existing good quality legacy components shall not be rewritten in Java for costs saving. Secondly, components of the system should sometimes be written by specialists who have their specific language of domain (e.g., Matlab). Thirdly, unless distributed, OSGi runs the application in a single virtual machine, i.e., every bundle shares the same memory area, so the lowest quality bundle determines the quality of the entire system. Consequently, they choose D-Bus (Desktop Bus) which is an inter-process communication solution with the following claimed properties: language independence and efficiency (however, no quantitative results is given for that case). Hence, components become separate processes directly managed by the operating system benefiting from its RT properties if available. D-Bus is heavily used in GNU/Linux both in desktop and system applications, and it is also ported to other operating systems including Windows. There exists bindings for a wide range of programming languages. Furthermore, it is also possible to connect several D-Buses running on different computers making distributed computing possible, which becomes in competition with initiatives like DOSGi [9] (Distributed OSGi). As a result, a D-Bus-based toolkit provides two new characteristics: language independence and component separation (see Section IV-B).

As a result, although these approaches propose complete solutions for component connection abstract and concrete model, none considers applying it to the widely used OSGi concrete model and perform a quantitative study of its incremental extension and provide a systematic methodology to score existing solutions to ensure the most adequate solutions rise to the top, which is the focus of this paper.

III. Constrained model of components

To present our connector model, we introduce in this section the model of our component system. We abstract it from any specific notation and represent it as a directed graph as in [6].

Definition 1. (Component graph) *A component graph $\mathcal{G}$ is a set of vertices V and directed edges E, noted $\mathcal{G} = (V, E)$. A vertex v has the attribute v.**typ**, a type. An edge e from v_1 to v_2 of V is written as $e = (v_1, v_2, l)$ where l is a label.*

Each vertex has a type which depends on the underlying system, i.e., communication bus or programming language.

Definition 2. (Input and Output) *Let $\mathcal{G} = (V, E)$ be a directed graph. For each vertex $v \in V$, its inputs are labels $I_1, \ldots, I_m$ noted v.**I** such that $\forall k \in [1, m], (v_k, v, I_k) \in E$. Similarly, its outputs are $O_1, \ldots, O_n$, noted v.**O** such that $\forall k \in [1, n], (v, v'_k, O_k) \in E$.*

In this model, a software connector is simply represented by a vertex as a component.

A. Updating state variables

The state variables, as introduced in Section I for solving heterogeneity problems, are represented by a set $\mathcal{S}$ where each element $\sigma \in \mathcal{S}$ has the attributes σ.**sta**, its state, σ.**id**, its identifier, and σ.**typ**, its type.

To update state variables, we define a fixed and contextual common message structure. The fixed part contains the *time stamp* of the message, the *type* of the associated state variable, a *regular expression (regexp) id* matching one or several state variable ids, and the *state*. The contextual part is a key-value *map* function on contextual information, such as the unit of the state, etc. Special connectors implementing the hardware abstraction layer are exhibited from the component graph. We call them HAC connectors (Definition 3).

Definition 3. (HAC) *Let $\mathcal{G} = (V, E)$ be a directed graph. A hardware abstraction connector (HAC) is a vertex $\gamma \in V$ such that for all $\pi \in \{\boldsymbol{I}, \boldsymbol{O}\}$, $\gamma.\pi = \gamma$ has the attributes **ts** (time stamp), **sta** (state), **re** (regexp id), **typ** (type), and **m** (key-value map).*

HAC connectors translate a regexp id from its input into a constant string id. On the one hand, as collected data are related to physical events, i.e., measurements of quantities (temperature, energy consumption, etc.) or notifications of state change (door opening, motion detection, etc.), the input regexp id is then reduced to a constant string representing the id of the state variable (e.g., *living-room.light.ceiling*). On the other hand, a sub-assembly of components should compute the matching of the state variable handling the action and translate the command to the proper communication frames. It allows us to specify the target using pattern matching rules, such as *living-room.light.** (id is supposed to follow a predefined format), which will alter state variables matching the rule receiving the appropriate command.

Property 1. (HAC invariant) *For all HAC noted γ in a component graph, the state of γ reflects the state of one or several state variables according to the matching of γ's regexp, i.e., for all HAC $\gamma \in V$ and $\sigma \in \mathcal{S}$, if σ.**typ** $= \gamma$.**typ** and σ.**id** $\cong \gamma$.**re**, then σ.**sta** $= \gamma$.**sta** where the symbol '$\cong$' stands for a regexp matching function.*

Definition 1 highlights the flexibility benefited from HAC connectors to target multiple state variables. Such an approach of the communication between the building blocks of the application allows us to decouple component implementations. We presented above the model of the toolkit and its properties. We propose in the next paragraphs one way of applying constraints to express a notion of performance.

B. Soft real-time constraints

Soft RT constraints impose that system response time should be adequate in average. At the connector level, this means that its delays should be compatible with these constraints. Thus, we propose to work with the ratio τ (called *performance*) of the duration noted d_C of a function call in C language as our yardstick reference to be independent of the CPU (central processing unit) load, i.e., the time between the function is invoked in a source component and the execution of its first instruction in a target component, over its own duration. We choose C for its widespread use as a simple procedural abstraction. This ratio is computed experimentally (see Section IV).

Definition 4. (Connector performance) *The connector performance τ of a connector type v.**typ** is a function $\tau(v.\textbf{typ}) = \frac{d_C}{d}$ scoring its performance between 0 (low) and 1 (high performance) where d is the computation time of v.**typ**.*

To compute the overall performance of the connectors of the system, we propose to take the average sum of all connector performances on the path of the predominant control flows. We do not seek to provide a predictive calculus for a given application, but rather a metric comparison between the different systems, achieved by a simple arithmetic average without weighted control flow.

Definition 5. (Performance of system) *The performance P of a set of n types of connector $(t_1, \ldots, t_n)$ and control-flow paths $\mathcal{F}$ is:*

$$P = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \left(\frac{1}{\sum_{j \in [1,n]} k_{f,j}} \sum_{i \in [1,n]} k_{f,i}.\tau(t_i) \right) \quad (1)$$

where $k_{f,i}$ is the number of times the path f traverses a connector of type t_i and $|\mathcal{F}|$, the cardinal of $\mathcal{F}$.

We built a model with a global performance function (Equation 1) which purpose is to provide a way to score the performance of a middleware. We show in the next section how to apply it to evaluate development solutions.

IV. D-BUS AS AN OSGI MIDDLEWARE EXTENSION

A. Connectors in iPOJO/OSGi

Apache Felix was selected as the OSGi implementation and iPOJO [10] (injected Plain Old Java Object) is used on top of it to provide an extensible and declarative component-oriented model. Thus, additional "handlers", i.e., *connectors*, can be implemented by means of extensions of the *PrimitiveHandler* class to intercept access to attributes and method invocations of *pojos*, i.e., functional codes. The first requirement in our projects was to specify service filters in a configuration file that selected the components associated with a required service. The existing connector *Require handler* of iPOJO proposes only a design-time service filter. We created a new handler (called *VariableRequire*) based on the latter to read a configurable property initialized with a given configuration file before adding filtering information during the assignment of a component. Secondly, we created another handler called *RegexpSubscriber* extending the event *Subscriber* of iPOJO to listen to broadcasted OSGi events. It provided a regexp HAC connector matching the OSGi topic as a complementary information instead of the internal OSGi specific filter. We do not detail the implementation of both connectors in this paper, but we focus on scoring them to provide a quantitative analysis of the system performance. We measured the C method call duration $d_C = 7\mu$s on average on Raspberry Pi (240 measures during 4h on low CPU load $< 1\%$) by using *gettime()* function. Two control flows are implemented. The first one writes sensor values into the database and traverses two connector types: 1 HAC (type t_0) and 2 iPOJO calls (type t_1). The second one logs battery level traversing the same connector types. Performance computation is shown in Table I.

B. Towards D-Bus extension

We introduced D-Bus in our architecture as in [7], in particular, by incrementally replacing the OSGi middleware. The idea was to gradually put the language-agnostic D-Bus (or other effective middleware) at the center of the communication between software components (see Figure 2) and observe the evolution in performance.

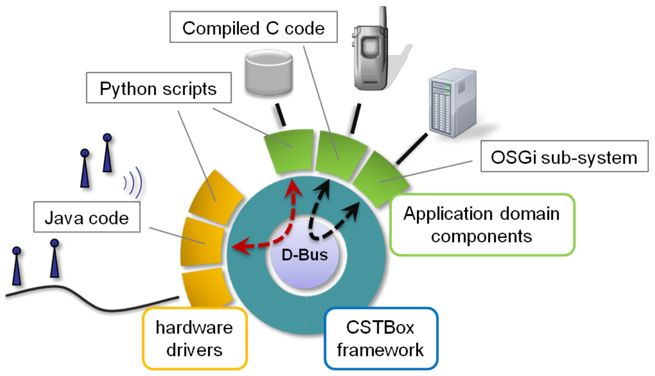

Figure 2. The operation principle of the CSTBox abstraction mechanism

D-Bus implementation requires a single process per bus entity, manages the process interface registration, and keeps overlaps in naming coherent by queuing bus requests. It also manages the life-cycle of processes. We have implemented the same sub-assembly as in Section IV-A, but in Python and D-Bus. The control flows are the same and traverse 2 types of connectors: 1 HAC (t'_0) and 2 Python calls (t'_1).

Table I
OSGI/IPOJO AND D-BUS PERFORMANCE COMPARISON

1 HAC (RegexpSubscriber OSGi)	$\tau(t_0) = \frac{7\mu s}{46000\mu s} \simeq 0.02\%$
2 iPOJO service call connectors	$\tau(t_1) = \frac{7\mu s}{13749\mu s} \simeq 0.05\%$

Performance of OSGi: $P_{\text{OSGi}} = \frac{1}{3}\left(\tau(t_0) + 2\tau(t_1)\right) \simeq \mathbf{0.04\%}$

1 HAC (CSTBoxEvent D-Bus)	$\tau(t'_0) = \frac{7\mu s}{4061\mu s} \simeq 0.17\%$
2 Python method call connectors	$\tau(t'_1) = \frac{7\mu s}{31\mu s} \simeq 22.58\%$

Performance of D-Bus: $P_{\text{D-Bus}} = \frac{1}{3}\left(\tau(t'_0) + 2\tau(t'_1)\right) \simeq \mathbf{15\%}$

Table I qualitatively shows that the performance of D-Bus is greater than OSGi's for this simple type of application architecture. This approach provides us with a tool to evaluate engineering choices and converge to suitable solutions, even though it still needs more experiments and validations.

V. Conclusion

The work described in this paper is derived from our experience designing, installing, and running experimentation for sensor network management and data processing unit engineering as an OSGi component toolkit. The design/verification flow is currently implemented as a helper tool computing performance from architecture description independently of the networks of equipments. Our toolkit provides and maintains a rich interfacing system, i.e., set of connectors between components. We worked at extracting the practical advantages from experiments and we capitalized on it by generalizing the connector model and evaluation method. As perspectives, we are considering achieving a complete experimental model of performance and the design of RT connectors and their certification by means of program proof in suited languages [11] to ensure safety property.

References

[1] N. Zouba et al., "A computer system to monitor older adults at home: Preliminary results," Gerontechnology Journal, vol. 8, no. 3, July 2009, pp. 129–139.

[2] SIGAAL Consortium, "SIGAAL Project," http://sigaal.org, [retrieved: June, 2013].

[3] A. Flamand and N. Roudil, "Domestic energy consumption: Inhabitants practices, rationality and motivation," in Int. Scientific Conf. on Sustainable Consumption - Towards Action and Impact, Abstract, Hamburg, Germany, Nov. 2011, p. 146.

[4] A. Zarli, E. Pascual, and D. Cheung, "Information and communication technology for intelligent and integrated control in buildings: Current developments and future research," in 27th Int. Conf. on Applications in IT in the AEC Industry, CIB, 2010 [published online, 10 pages].

[5] C. A. Mattmann, D. Woollard, and N. Medvidovic, "Exploiting connector knowledge to efficiently disseminate highly voluminous data sets," in 3rd Int. Workshop on Sharing and reusing architectural knowledge. New York, NY, USA: ACM, 2008, pp. 37–40.

[6] S. Fathallah, S. Lavirotte, J.-Y. Tigli, G. Rey, and M. Riveill, "The dynamic composition of independent adaptations including interferences management," in 7th Int. Conf. on Soft. Eng. Advances, Lisbon, Portugal, Nov. 2012, pp. 678–684.

[7] P. Hanák et al., "System architecture for home health and patient activity monitoring," in 5th European IFMBE Conf., 2011, pp. 945–948.

[8] T. Kovácsházy, G. Fodor, and C. B. Seres, "A distributed power consumption measurement system and its applications," in 12th Int. Carpathian Control Conf., May 2011, pp. 224–229.

[9] S. Mohammed, D. Servos, and J. Fiaidhi, "Developing a secure distributed OSGi cloud computing infrastructure for sharing health records," in 2nd Int. Conf. on Autonomous and Intelligent Systems. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 241–252.

[10] C. Escoffier and R. S. Hall, "Dynamically adaptable applications with iPOJO service components," in 6th Int. Symp. on Software Composition. Braga, Portugal: Springer-Verlag, July 2007, pp. 113–128.

[11] P. Letouzey, "Extraction in Coq, an overview," in Logic and Theory of Algorithms, 4th Conf. on Computability in Europe, A. Beckmann, C. Dimitracopoulos, and B. Löwe, Eds., vol. 5028. Springer-Verlag, 2008, pp. 359–369.