SOA Real-time System Development An Automotive Case Study

Cuong M. Tran, Kung-Kiu Lau, Simone Di Cola
School of Computer Science, The University of Manchester
Manchester M13 9PL, United Kingdom
ctran,kkl,dicolas@cs.manchester.ac.uk

***Abstract*—Real-time systems have become increasingly complex and therefore requires scalable development solutions. Combining component-based and SOA paradigms can be one of such solutions by enforcing loosely coupled reusable components and suitable composition mechanism to construct systems. In this paper we demonstrate our solution using the X-MAN component model and its real-time extension on an industrial case study in automotive.**

***Keywords*—component architectures, service computing, real-time systems**

I. Introduction

Real-time systems have become increasingly complex and therefore requires scalable development solutions. Component-based development promotes systematic reuse of pre-existed components. On the other hand SOA paradigm fosters loosely coupled and self-contained functional units. The combination of the two approaches provides a solution to the above problem. In this paper we demonstrate this combination using the X-MAN component model and its real-time extension applied to an industrial case study in the automotive domain.

II. Case Study

Provided by IXION Industry and Aerospace[1], the case study is the navigation system part of a highly automated driving vehicle control system. In brief, the system receives data from in-car cameras and sensors, determines its location and surrounding obstacles, and visualise them on the map. The system requires concurrency and strict timing to efficiently handle data and perform location calculation and image processing.

III. Our Approach

A. The X-MAN Component Model

In the X-MAN component model [9], [10] (Fig. 1), there are two basic entities: (i) *components*, and (ii) *composition connectors*. Components are units of design with behaviour, which is exposed by its provided services. They can be atomic or composite. An atomic component (Fig. 1(a)) is a unit of composition and computation. Its computation unit (CU) self-contains implementation of the services it exposes via invocation connector (IC). Components are composed via composition connectors (Fig. 1(b)) into composite components (Fig. 1(c))). X-MAN provides four basic composition connectors, which are *Sequencer*, *Selector*, *Aggregator*, and *Par*.

[1]http://ixion.es

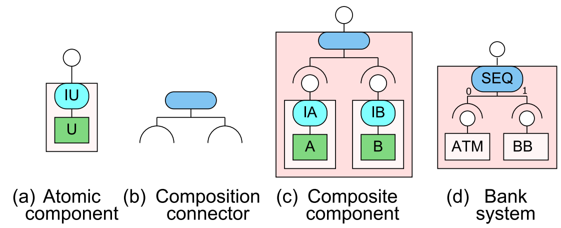

Fig. 1. The X-MAN component model.

Sequencer provides sequencing, *Selector* offers branching, *Aggregator* provides a faade, whilst *Par* enables parallel invocation. In a composite component sub-components do not call each other. Instead, the composition connector coordinates the sub-components' execution. For instance, in the bank system in Fig. 1(d), the sequencer *SEQ* first calls *ATM* to get customer inputs and then calls the bank branch *BB* to handle the inputs.

Behaviour of single components can be adapted by *adaptor* connectors. They are *Loop* and *Guard*. *Loop* provides iteration, while *Guard* gating.

The execution semantics for X-MAN is control-driven, with explicit data routing. The latter cab be *horizontal* or *vertical*. *Horizontal* data routing is between sub-components within a composite component. *Vertical* data routing is data propagation between the interface of a composite component and its sub-components.

B. Real-time extension

In order to support real-time system development, we extended the X-MAN component model with the concept of real-time view to capture timing concerns. Within this view, periodic computations are implemented as recurring invocations of component services by using a *Timer* loop[2] with a *period* attribute. In addition, *Constraints* are to capture deadlines and priorities, whilst *Data elements* shared resources.

This extension is fully implemented in the X-MAN II toolset[3] [6] which includes a graphical designer and assembler, and a code generator.

IV. System construction

After the analysis of the case study, we construct the system bottom up starting from atomic components. An example of

[2]*Timer* loop is an adapter connector.
[3]X-MAN II toolset - Availableathttp://www.mub.eps.manchester.ac.uk/xman/

such is depicted in Fig. 2. The EKF component, which implements the Extended Kalman filter[4], exposes a service called *CalcEKF*, which takes six inputs (*GPS*, *IMU*, *CAN*, *car_data*, *ekf_vars_data*,*headingMap*) and produces one output (*ekfOut*). The behaviour of this computation is implemented in C language within the computation unit *CU*.

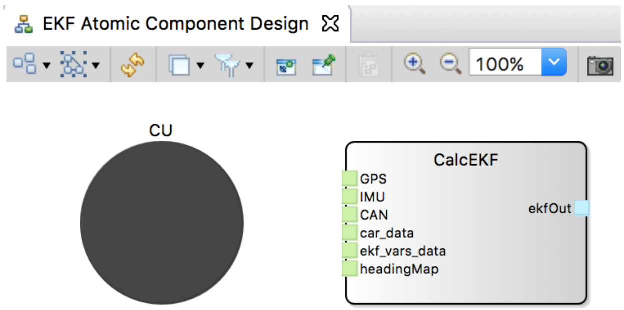

Fig. 2. EKF atomic component.

Once validated, components can then be stored in a repository as in Fig. 3. Thereon, they can be are retrieved to

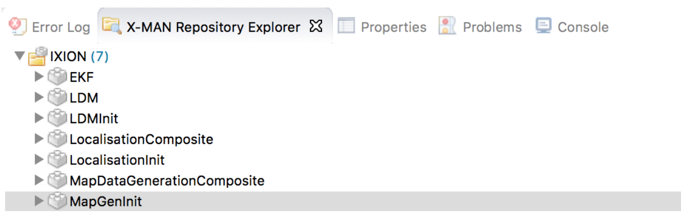

Fig. 3. Component repository.

construct new (composite) components. An example of such is depicted in Fig. 4. We reused two previously built components `QuadrantDetection` and `MapGeneration` by composing them with two connectors which are *SEQ1* and *GRD0*. The connectors together first triggers obstacle detection service *Detect*, then conditionally triggers map generation service *GenerateMap*. This sequence of computation is exported via a new service called *Detect_GenerateMap*. Clearly, this demonstrates that we hierarchically constructed systems. It also shows that every step of construction gives us a new component with new services with complex behaviours, realised by the involved components and connectors.

Similarly, in Fig.5 we use the *Par0* connector to compose `LDMInit`, `LocalisationInit` and `MapGenInit` to concurrently initialise three main sub-systems of the system. The new service *Detect_Generate_Map* acts the entry point of the navigation system.

The real-time view of the system is depicted in Fig. 6. Each of the three components `Localisation`, `MapGeneration` and `LDM` is adapted by a *Timer* loop (e.g. *MapGenTimer*) and a constraint (e.g. *Localisation Constraint*).

[4]https://en.wikipedia.org/wiki/Extended_Kalman_filter

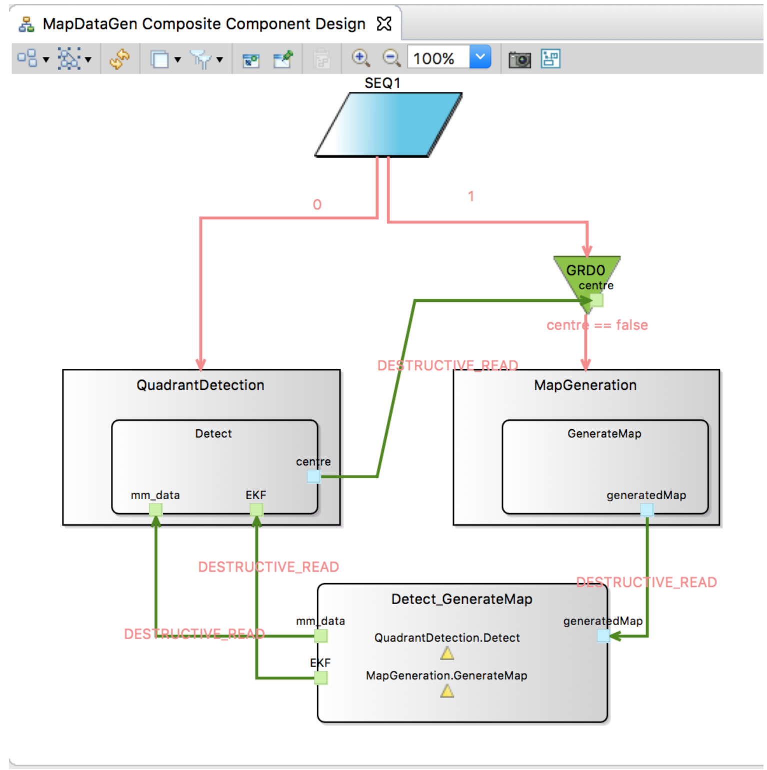

Fig. 4. MapDataGeneration composite component.

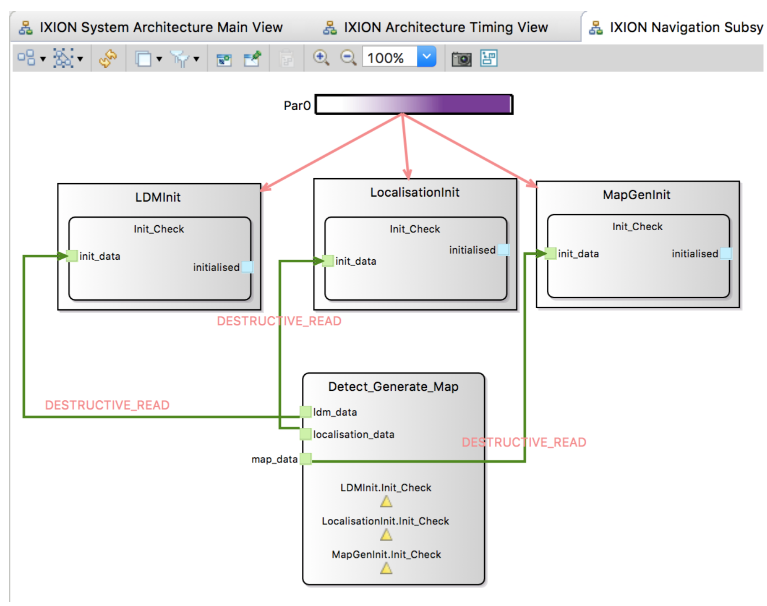

Fig. 5. Concurrent sub-systems.

Shared resources, such as *shared_mm_data*, are also specified. Finally, the system is completed by aggregating the composite component in Fig. 5 with the timing view in Fig. 6. The result is illustrated in Fig. 7. The system is executed by calling the service *Detect_Generate_Map*, which initialise and enable the real-time adapted components.

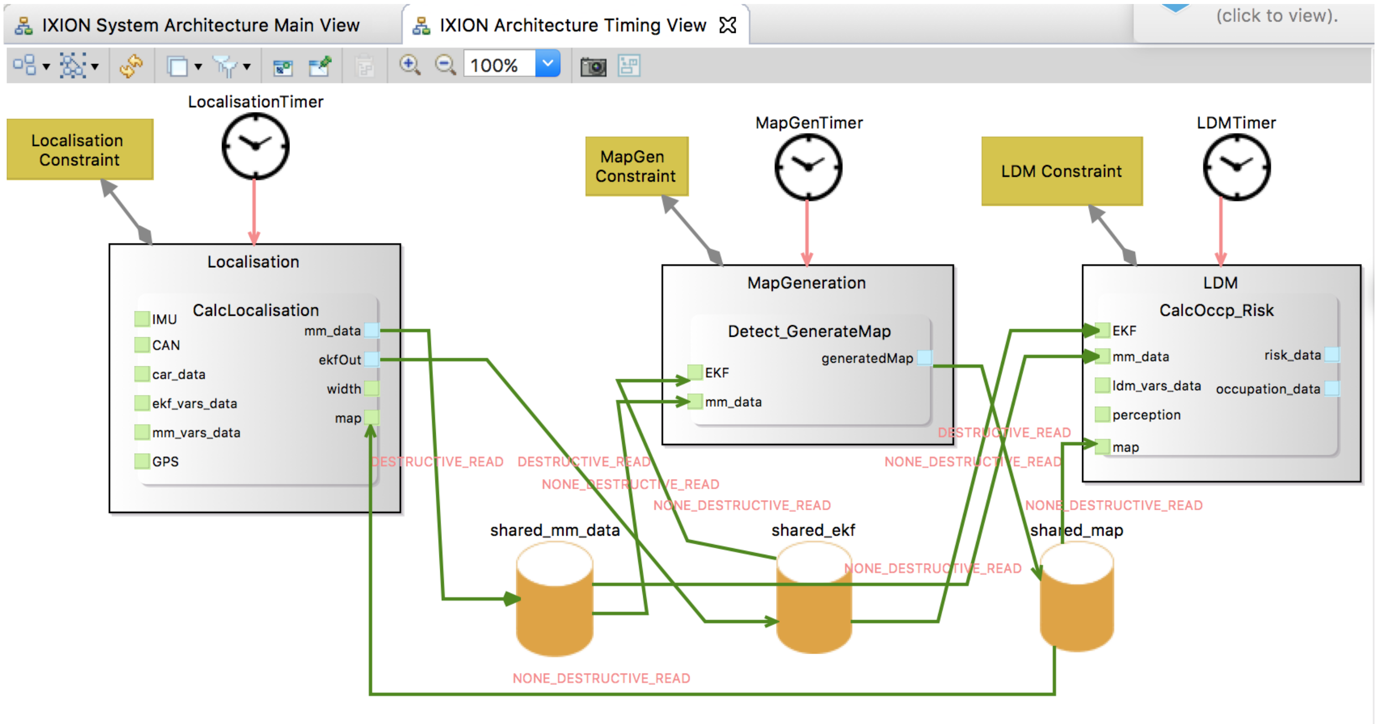

Fig. 6. MapDataGeneration composite component.

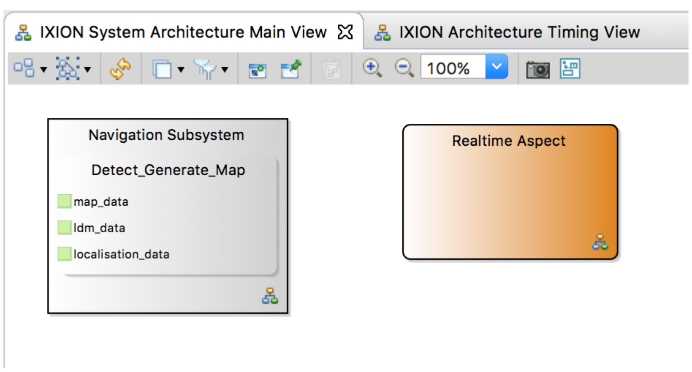

Fig. 7. Navigation system.

V. Related work

There have been many approaches in constructing SOA architectures such as BPEL [13], JOpera [12], SCA [11]. However, they are restricted to the domain of informative systems having web services as the implementation. Support for real-time systems is completely lacking. In contrast, X-MAN supports both SOA architectures and real-time.

Real-time system development over the years has received a number of solutions in the form of component models (ProCom [4], UML with MARTE [2]), synchronous languages (Esterel [3], Lustre [8], SIGNAL [7]), as well as specialised programming languages with real-time extensions (Ada 95 [5], Java RTS [1]).

ProCom has two layers: (i) ProSave to model passive and sequential components; (ii) ProSys to model active components and systems. Compared to X-MAN, ProCom does not make use of pre-defined composition operators (for coordination) which makes the composition more ad hoc and the control logic implicit. ProCom does not have a support tool either. MARTE is a UML profile which enables annotating UML diagrams with real-time constraints. As UML components are objects which are tightly coupled an fine grained. As a result they are not suitable for SOA.

Synchronous languages abstract the concept of time as logical ticks. In every tick, computation and data communication perform instantaneously. Components in these languages are function blocks, or state machines, which are wired up using data paths. While support on real-time and reactive system construction is strong, the components are not service oriented and hence support of SOA does not exist. This would make systems constructed in these languages (e.g. Esterel) difficult to integrate with other (potentially non-) real-time systems through a standard interface.

Ada 95 with real-time annex and Java RTS are also capable of developing real-time systems. Components in these languages are 'tasks'. However, they are essentially programming languages and therefore do not offer the right abstraction for complex system modelling. Moreover, SOA is not directly possible with these languages.

VI. Conclusion

In this paper, we have presented our solution to developing an SOA real-time system using our X-MAN component model and its extension. The construction was performed using our tool-set. In the future, we will improve the code generator to enhance the quality of generated code. Furthermore, we are planning to implement a model transformer to generate task model for timing analysis and validation.

Acknowledgment

* We would like to thank IXION Industry & Aerospace Spain and Dr. Jorge Villagra in particular for providing us the case study as well as contributing feedbacks to our approach.

** Research leading to these results has received funding from the EU ARTEMIS Joint Undertaking under grant agreement no. 621429 (project EMC2) and from the Technology Transfer Board (TSB) on behalf of the Department for Business, Innovation & Skills, UK.

References

[1] Sun java^{TM} real-time system 2.2 update 1 technical documentation, April 2010.

[2] The uml profile for marte: Modeling and analysis of real-time and embedded systems, 2015.

[3] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[4] Tomas Bures, Jan Carlson, Ivica Crnkovic, Severine Sentilles, and Aneta Vulgarakis. *ProCom — the Progress Component Model*. Malardalen University, Vasteras, Sweden, 2010.

[5] Alan Burns and Andrew J. Wellings. *Real-time Systems and Programming Languages: Ada 95, Real-time Java, and Real-time POSIX*. Pearson Education, 2001.

[6] Simone Di Cola, Cuong Tran, and Kung-Kiu Lau. A graphical tool for model-driven development using components and services. In *Proceedings of 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA) 2015*, pages 181–182, 2015.

[7] Abdoulaye Gamati. *Designing Embedded Systems with the SIGNAL Programming Language: Synchronous, Reactive Specification*. Springer, 2009.

[8] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow language lustre. In *Proceedings of the IEEE*, volume 79, pages 1305 – 1320, 1991.

[9] N. He, D. Kroening, T. Wahl, K.-K. Lau, F. Taweel, C. Tran, P. Rümmer, and S. Sharma. Component-based design and verification in X-MAN. In *Proc. Embedded Real Time Software and Systems*, 2012.

[10] K.-K. Lau, M. Pantel, D. Chen, M. Persson adn M. Törngren, and C. Tran. Component-based development. In A. Rajan and T. Wahl, editors, *CESAR – Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*, chapter 5, pages 179–212. Springer-Verlag Wien, 2013.

[11] Jim Marino and Michael Rowley. Understanding sca, 2009.

[12] Cesare Pautasso. Composing restful services with jopera. In *8th International Conference on Software Composition (SC)*, volume 5634, pages 142–159, Zurich, Switzerland, 2009. Springer Berlin Heidelberg.

[13] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005.