Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Using a data-centric event-driven architecture approach in the integration of real-time systems at DTP2

Janne Tuominen*, Mikko Viinikainen, Pekka Alho, Jouni Mattila

Tampere University of Technology, Department of Intelligent Hydraulics and Automation, Tampere, Finland

ARTICLE INFO

Article history:
Received 13 September 2013
Received in revised form 4 April 2014
Accepted 16 April 2014
Available online 14 May 2014

Keywords:
DTP2
Middleware
Information model
Communication
Real-time control system

ABSTRACT

Integration of heterogeneous and distributed systems is a challenging task, because they might be running on different platforms and written with different implementation languages by multiple organizations. Data-centricity and event-driven architecture (EDA) are concepts that help to implement versatile and well-scaling distributed systems.

This paper focuses on the implementation of inter-subsystem communication in a prototype distributed remote handling control system developed at Divertor Test Platform 2 (DTP2). The control system consists of a variety of heterogeneous subsystems, including a client–server web application and hard real-time controllers. A standardized middleware solution (Data Distribution Services (DDS)) that supports a data-centric EDA approach is used to integrate the system.

One of the greatest challenges in integrating a system with a data-centric EDA approach is in defining the global data space model. The selected middleware is currently only used for non-deterministic communication. For future application, we evaluated the performance of point-to-point communication with and without the presence of additional network load to ensure applicability to real-time systems. We found that, under certain limitations, the middleware can be used for soft real-time communication. Hard real-time use will require more validation with a more suitable environment.

1. Introduction

Integration of heterogeneous and distributed systems is a challenging task, because they might be running on different platforms and written with different implementation languages by multiple organizations. Service-oriented architecture (SOA) is an architectural design pattern that has been developed to deal with this problem in enterprise information systems, typically using Web services and XML based protocols. This approach has some downsides when applied to deterministic systems, e.g. uncertain communication latencies due to time coupling between services, increased development complexity and, in some cases, a possible single-point-of-failure caused by a central message broker. A better fit for deterministic systems is a data-centric event-driven architecture (EDA) that focuses on providing a global data space instead of services, and decouples the communicating elements with respect to time.

This paper focuses on the implementation of inter-subsystem communication in a prototype distributed remote handling (RH) control system developed at Divertor Test Platform 2 (DTP2), specifically on issues related to performance and interfaces. The DTP2 system uses the Object Management Group's (OMG) standard specification for Data Distribution Service for Real-Time Systems (DDS) for ensuring communications interoperability. DDS lacks a centralized broker, thereby avoiding a single-point-of-failure, and is also designed with the real-time requirements of control applications in mind.

One of the greatest challenges in integrating a system with a data-centric EDA approach is in defining the global data space model, and we will present information on application to the current DTP2 system. At this point in time, DDS is only used to implement non-deterministic communication channels. For future application, we will also evaluate the performance of point-to-point communication with and without the presence of additional network load to ensure applicability to real-time systems.

2. Divertor Test Platform 2 remote handling system

A key ITER maintenance activity is the replacement of the divertor cassettes, which is scheduled to be performed after every 3–4 years of plasma operations. DTP2 is a full scale physical test facility located at Tampere, Finland. The purpose of DTP2 is

* Corresponding author. Tel.: +358 401981221.
E-mail address: janne.m.tuominen@tut.fi (J. Tuominen).

http://dx.doi.org/10.1016/j.fusengdes.2014.04.040
0920-3796/© 2014 Elsevier B.V. All rights reserved.

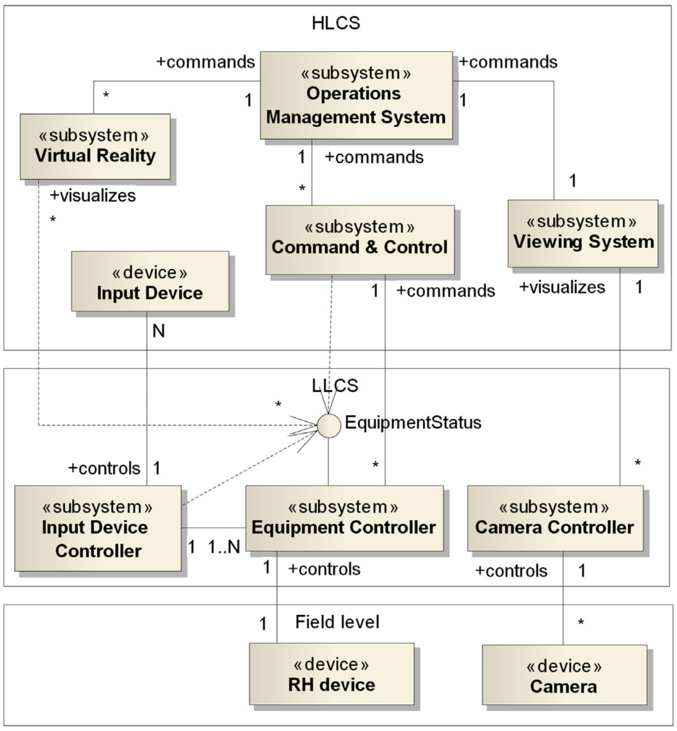
HLCS
«subsystem» Operations Management System
«subsystem» Virtual Reality
«subsystem» Command & Control
«subsystem» Viewing System
«device» Input Device
+commands
+visualizes
LLCS
EquipmentStatus
«subsystem» Input Device Controller
«subsystem» Equipment Controller
«subsystem» Camera Controller
+controls
Field level
«device» RH device
«device» Camera

Fig. 1. DTP2 control system.

to develop and validate RH equipment designs with prototypes closely replicating those proposed for ITER, see e.g. [1].

The ITER RH environment will consist of a large number of software systems that must cooperate with each other and be coordinated by a higher-level system. RH control system development has been performed at DTP2 for enabling divertor cassette replacement procedure testing, as well as validating control system software solutions for Fusion for Energy (F4E) and ITER.

General requirements for the control systems, according to [2], are high reliability and high flexibility. The system should be capable of recovering from unforeseen situations and failures. As explained in more detail in [3], the control system at DTP2 consists of a heterogeneous collection of subsystems, implemented on different operating systems (Windows, Linux, and LabVIEW Real-Time) and with different programming languages (C++, PHP, LabVIEW G language, etc.). The complete system is fairly extensive in scope, and it is typical that one programming environment is more suitable than others for one subsystem, area of expertise and development team, hence the heterogeneity. Most of these systems are integrated by using a standardized middleware over switched Ethernet.

As shown in Fig. 1, the system is divided into three levels: High-Level Control System (HLCS), Low-Level Control System (LLCS) and Field-level equipment. The HLCS consists of subsystems that provide graphical user interfaces for the system operators. These subsystems command the LLCS subsystems, which are used for abstracting the actual hardware from the HLCS subsystems. For example, the EC subsystem executes in hard real-time and implements the low-level closed-loop controls and kinematic calculations that are needed to drive the RH device.

3. Adopted technologies

3.1. Event-driven architecture

As explained e.g. in [4], in event-driven architecture, notable events are detected and event messages are generated. The event messages are processed by an event processing engine and then disseminated to all interested parties, who can then evaluate the events and optionally take action. The event processing engine can also optionally invoke services, start processes or store the event messages.

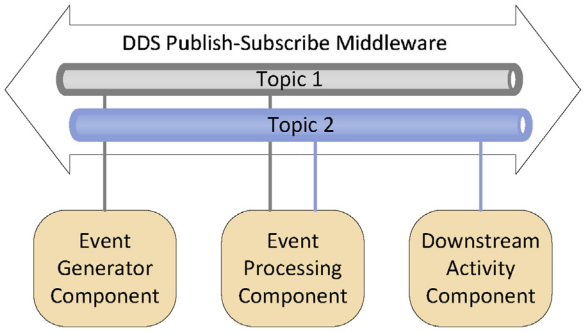
DDS Publish-Subscribe Middleware
Topic 1
Topic 2
Event Generator Component
Event Processing Component
Downstream Activity Component

Fig. 2. Event-driven architecture with DDS.

The creators of the events (event generators) only know that the events have transpired and that the event has been sent out to the event processing engine. They therefore do not know if any actions were taken to respond to the event, or how many (if any) parties were interested in the event. This encourages loose coupling between architectural elements and is best suited for asynchronous flows of information.

3.2. Middleware

The publish-subscribe oriented Data Distribution Service (DDS) middleware standard [5] was chosen for system integration due to its performance, scalability and user-adjustable Quality of Service (QoS) policies [3].

Fig. 2 depicts the use of DDS in an event-driven architecture. The event processing engine discussed in Section 3.1 is replaced by a dedicated event processing component [6]. DDS relays messages between components in “pipelines” called topics. All published data is made available to all components in a global data space that can be sectioned into domains to control data visibility.

The DDS implementations support automatic discovery of communicating nodes, do not need a central broker node, support anything between one-to-one and many-to-many connections and are designed with real-time capabilities in mind. DDS is referred to as a data-centric publish-subscribe middleware due its strong emphasis on data modeling [6].

4. Information models

Interfaces are the externally visible parts of architectural elements, such as subsystems, components, active objects and classes. These interfaces should be general enough and stable in order to promote low coupling and reusability. Information models are valuable concepts to use for building stable interfaces.

An information model defines the structure and semantics of data objects that are passed between communicating elements [7]. An interface consists of function or subroutine calls and data types. Some of those data types may refer to information models. The final interface of an element consists of specific versions of an interface definition and an information model.

The main benefits of using information models are the possibility of reuse in several interfaces, more stable methods in interfaces, the ability for designers to handle concepts as separate entities, and simplified handling and storing of data objects (as the data is handled and understood in a similar manner across the different elements). It is generally not feasible to apply information