CONFIGURATION MODEL FOR CONTROL KERNEL MIDDLEWARE BASED APPLICATIONS

Jose L. Beltrán, Lorena Calabuig, Eduardo Munera, José Simó, Jose-Luis Poza-Lujan
Institute of Control Systems and Industrial Computing
Polytechnic City of Innovation
Polytechnic University of Valencia, Spain
{jobelal2; locamo}@inf.upv.es, {emunera; jsimo; jopolu}@ai2.upv.es

Abstract

In this paper the configuration model for Control Kernel Middleware is presented. The configuration capabilities of the system have been defined in order to provide a clean interface for managing system elements. The development of a user interface that implements the model provides a more reliable and efficient way to perform configuration tasks. A communication system based on DCPS is used as base to achieve reliable communications through the distributed system. The communication structures for message exchange will be introduced. Finally some experiments for the validation of the tool will be presented.

Keywords: control kernel, configuration tool, communication system, distributed control system, embedded system.

1 INTRODUCTION

Distributed control systems are more and more used in complex architectures. It is common to find this systems in flight controllers, robots, autonomous vehicles, etc. where sensors, actuators and control devices are scattered. Typically, these systems do not usually provide friendly configuration interfaces so for the user it is difficult to make any modifications. Furthermore, it is usual that configuration files have considerable sizes so user is forced to know the structure in order to manage it. In this regard, human errors are common. A software tool which integrates configuration capabilities of the system provides transparency to the user, reduce errors and optimize system use.

The configuration model obtained for Control Kernel Middleware (CKM) [1][5][14] is presented in this work. The development of an user interface that implements the mentioned model allows to generate configuration files automatically. The submission of the setting are provided by a communication system based on DCPS which spread the information required using topics.

This paper is organized as follows: in section 2 a revision of the existing configuration tools in embedded systems is done. Section 3 presents the configuration model obtained for the Control Kernel Middleware. Section 4 will deal with the resulting configuration files. In section 5 is presented the graphic interface implemented from model. The communication system designed for broadcast information by topics is presented in section 6. Section 7 shows the obtained results in order to validate the tool. And finally in sections 8 and 9 the conclusions obtained and the future lines of work is shown.

2 RELATED WORK

The Control Kernel was presented as a solution to process control where the objective is to achieve transparency in the design and applications development of real-time control. Subsequently, their implementation as middleware [4][9] has provided a set of services and interfaces used for basic management of the components of a control system: *sensors*, *actuators* and *controllers*. This middleware has two implementations, the full version called *Full Middleware* (FCKM) [9] and the reduced version called *Tiny Middleware* (TCKM) [9] used in systems with limited resources.

The CKM has a multitude of configuration options in a similar way as in others middleware. To ease the design process to the user, these middleware have graphical interfaces to configure it. An example is *Choregraphe* [12], a program developed by Adelbaran Robotics for NAO robots. Choregraphe allows users to define robot behaviors without any knowledge of programming. A predefined behaviour can be easily configured through the use of a block diagram edition environment.

More extended and specific configuration tools can be found as [11] or *rxDeveloper* [8] which offers modification of nodes interactively. Furthermore, tools like POLCOMS [18] interface allows the visualization and real-time parameterization of the system data. This interface implements CORBA [10] for communication between client and server.

Some communication middleware that implement the publication/subscription model offer graphical interfaces [7] for variable modification. Taking as an example the above tools, a user interface

ISBN-13: 978-84-697-0589-6

has been designed to help the programmer in the CKM configuration task.

3 CONFIGURATION MODEL

Control Kernel Middleware has a set of services [9] responsible for managing the basic components of a control system: sensors, actuators and controllers. Each component has different configuration capabilities. It has been defined a configuration model based in xsd files [19] [2] which specify all the needed parameters for starting the middleware offered services and load the desired application. This model contains precise information about the definition of the required parameters to perform a service request and its configuration. In Fig. 1 can be observed a global representacion of the model.

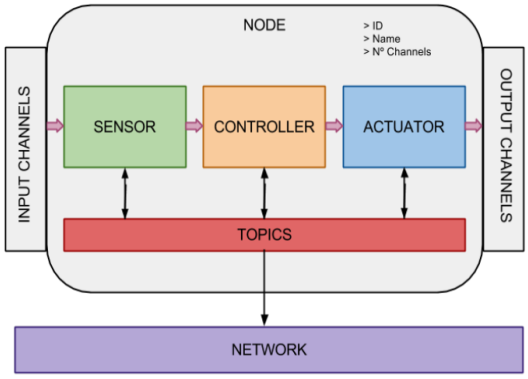

Figure 1: Configuration model of CKM

The system can be comprised of one or more nodes. In the distributed system, each node can have a combination of the components shown in the previous figure. All middleware components have been modeled. Below there is a brief description of each of them: nodes, sensors, actuators, controller, channels and communication mechanisms.

- The node is a logical component which has no associated specific functionality. Represents a physical element that contains the other components such as sensors, actuators, controllers, channels and topics.
- Sensors and actuators are logical components that connect the input and output channels of a node respectively. Both components define a predefined set of functions for the signal processing. The control engineer can use the defined functions or specify your own.
- The controller is a purely logical component. It is characterized by not having any associated hardware channel.
- The topcis are used for the dissemination of information between components. A topic is classified as internal or external. An internal topic is used within the scope of a node. Communications between different nodes are provided by external topics. This classification is only done conceptually. Internally, the middleware does not differentiate between the two types.
- Channels are considered as logical representations of a physical input/output element.

Hierarchical model allow to extract partial configuration from distinct modules and/or components. The model definition based on xsd files provides mechanisms to validate the configuration files. In Fig. 2 is presented the representation of a node by using XSD code.

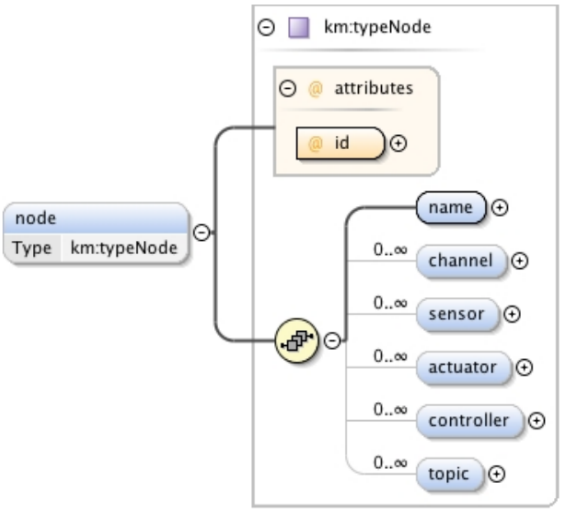

Figure 2: Representation of a node by using XSD code.

Once the model is specified in XSD code any application or user can validate the configuration files from the defined model.

4 CONFIGURATION FILES

In Control Kernel Middleware, the configuration is obtained through the use of xml files, who are generated from the model. In the same way as the model, xml files can describe the configuration both totally or partially. Every xml contains valid configurations in any case that is able to be validated itself against the model through standard xml/xsd validation mechanisms [6].

Once the configuration is disseminated through the communications system, the middleware is able to load this xml configuration files. Aflter loading and validating these files are performed

ISBN-13: 978-84-697-0589-6

the needed middleware calls for setting described configurations.

Although configuration files can be edited in manual mode using a text editor, and following the described criteria in the mode, in most occasions, this edition mode is very unfriendly and can easily lead to errors. For that reason it has been developed an user interface which allows the user to generate and modify configurations in a practical way, without the need of a deep known about the model.

5 DEVELOPMENT INTERFACE

In order to ease as much as possible the application design process, a user interface has been implemented. The interface allows the user to design the control application avoiding required real time (RT) programming, communications, etc. In Fig. 3 can be observed a capture of the configuration and edition application.

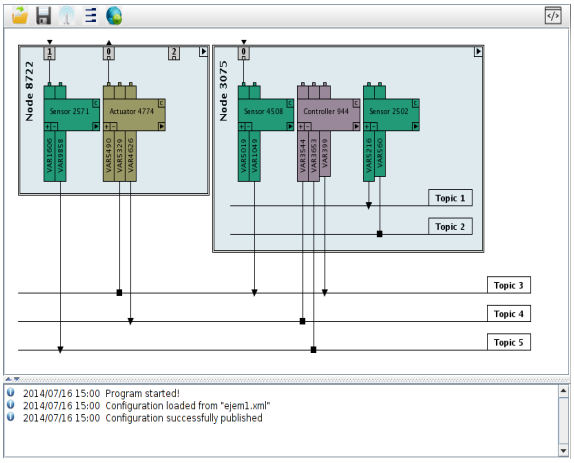

Figure 3: Configuration editor interface

Once the design is fulfilled, application generates xml files that will be used by the CKM to load the application. The settings are sent by using the communication system described in section 6. The application validates xml files generated from the defined model. In this sense, it has managed to design applications without the need to have knowledge about the structure of the model. The generated files can be edited from the same application or manually through a simple editor.

In addition, interface also allows to connect with the middleware at run time for handling the configuration dynamically, so in those cases that the user perform configuration changes during the execution, these changes can be stored within the rest of configuration files for future executions.

6 COMMUNICATION SYSTEM

Distributed system functioning is based on the possibility of the communication between different components. Communication system covers aspects of diverse complexity, whether it is connection and message addressing or message format to spread the content of that messages.

The user interface is responsible for sending the configurations to the rest of the distributed system. For this it is necessary to have a communication system that ensures reliable delivery of messages. It has been implemented a communication system based on Data Distribution Service (DDS) [3] model for this purpose.

DDS model defined in [16] is a specification for distributed systems that simplifies complex network programming and implements a publish/subscribe model for sending and receiving data, events and commands among the nodes. The DDS specification is divided into two layers:

- Data-Centric Publish-Subscribe (DCPS): The lower level that is targeted towards the efficient delivery of the proper information to the proper recipients.
- Data-Local Reconstruction Layer (DLRL): An optional higher level which is responsible for adapt the local applications data.

As in this article is pretended to illustrate the dissemination of necessary information for the configuration tool, it is only described DCPS communication model of DDS.

6.1 DCPS COMMUNICATION MODEL

DCPS model, explained in [3][13], is a communication model that provide the functionality required by an application based in publish/subscribe paradigm for data distribution using *topics*.

In this context, publisher are responsible for writing in a topic and subscriber reads that topic. As much publisher as subscriber are in charge of the communication between control system and communication system and are separated. This separation allows to organize and manage communications efficiently; this is the concept of a middleware.

6.2 BROADCAST CONFIGURATION

The communication system implemented handles connexions to spread messages to the network

ISBN-13: 978-84-697-0589-6

peers, so that a bidirectional connection between the different pairs of nodes is established.

All peers know the another peers connected so we are talking about a fully connected mesh topology. In this context, each peer publish its identity by a diffusion channel. In order to establish a pairing pattern of connections, each peer can only connect to another peer which GUID is greater.

The best way to communicate and synchronize peers in distributed systems is by message passing [15]. To enable the system to distinguish the type of message received, the following types are implemented:

- *Plain*, containing any kind of information.
- *TopicData*, containing the topic to manage data inside the DCPS service.
- *TopicLink*, containing a list of a topics to subscribe or unsubscribe.
- *SyncTime*, that is used to synchronize clocks of a different peers.
- *ProxyWrapper*, wrapping a message in another message in order to send to more than one port.

The user interface within the system is considered as a peer more. The configurations are broadcast through messages of type *TopicData*.

6.3 MESSAGE FORMAT

The communication system uses different types of message as mentioned in 6.2. Every message have a header and a payload, but also have a prologue (Fig. 4), that the system uses it to distinguish the different messages received. Prologue contains the type of a message (*Type*).

Other fields are also included in the prologue: *flag*, used to discard messages that are not successfully received; *size*, indicates the size of the message header and payload; and *source IP* and *port*, indicates the peer IP and port that sends the message. Figure 4 shows all the fields previously explained.

TopicData message have a specific header and payload as it is shown in Figure 4. In the message header, *time stamp* is measured in microseconds, *topic name size* is the size of the *topic name*. The *payload buffer* contains the data to be transmitted, in this case, the data to set the tool is in xml format.

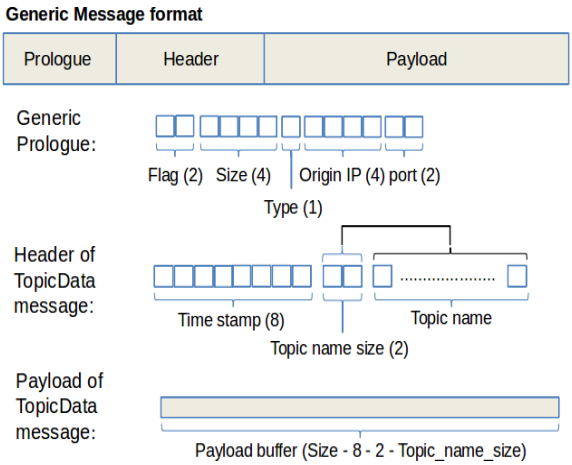

Figure 4: Message format for TopicData

7 EXPERIMENTAL RESULTS

Several experiments have been developed in order to validate the correct operation of the user interface. The goal is to verify that the configuration files are correct and the model was properly defined. For this purpose, a KertrolBot [17] robot is used. This robot is equipped with two infrared sensors and has Wi-Fi communications.

The scenario could be defined as shown in Fig. 5. It is a distributed system consisting of two nodes implementing a version of CKM. Node 1 is a personal computer with the FCKM version. In this node, the user interface is executed. Communication between nodes is done via the TCP protocol. Node 2 implements versions TCKM where the configurations are received and applied.

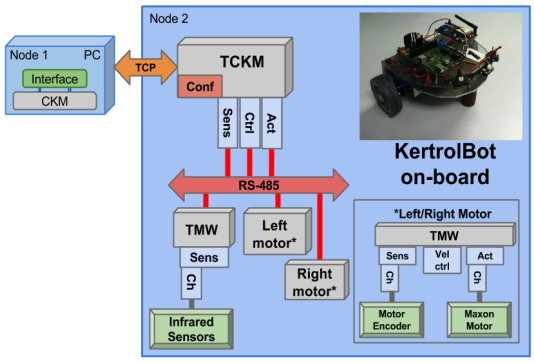

Figure 5: Experimental platform system overview

In each experiment, the robot moves through a room with obstacles which have been placed randomly. The idea is to observe how the robot acts on the environment based on the configurations applied in each moment. Developed experiments are described below:

- Experiment 1: The robot starts moving with

ISBN-13: 978-84-697-0589-6

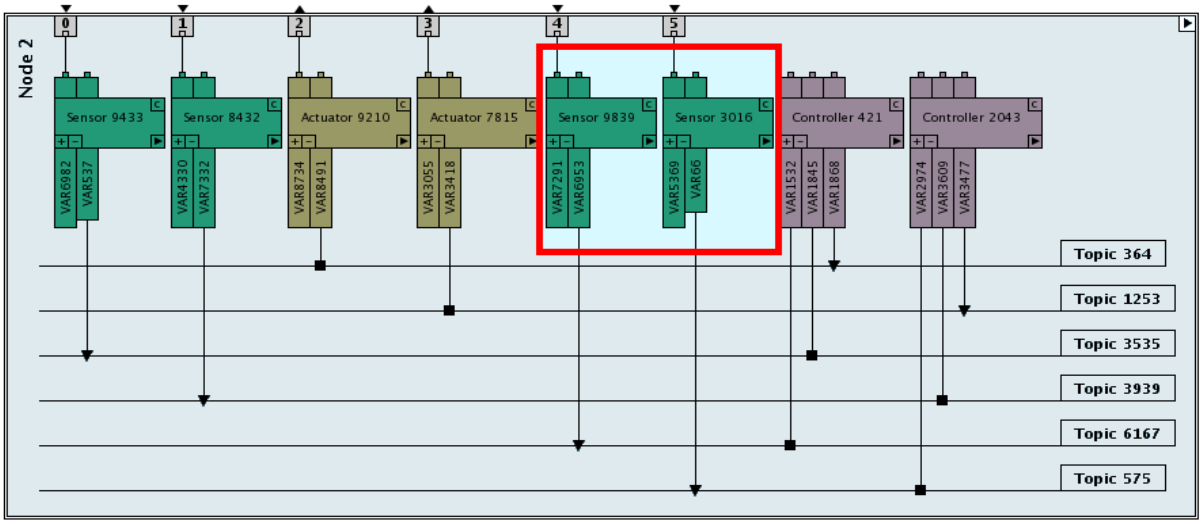

Figure 6: Configuration management tool showing the design of the experiments

disabled sensors. This behavior causes the robot collides with obstacles. Once in motion, the left sensor is enabled. The goal is to verify that the robot manages to avoid obstacles detected on the left side.

- Experiment 2: It is a continuation of the experiment 1. The robot starts moving with the configured behavior in the previous experiment. The objetive is to verify that the robot is able to avoid obstacles on both sides.
- Experiment 3: Like the previous case, the robot starts its motion from the last set configuration. In this case the goal is to disable both sensors and check that the robot is not able to detect obstacles in its environment.

In all experiments it has been proved as the configurations influence the control system. On the other hand, it also verifies that the configurations are applied at runtime. In Fig. 6 can be observed the designed application for the experiments.

8 CONCLUSIONS

The configuration model of the Control Kernel Middleware has been defined. All the basic components of a control system have been included. From this model it is possible to generate full configuration files with all the capabilities of the middleware configuration currently supported. A hierarchical structure allows to extract partial configuration from distinct modules and/or components.

Using XSD schema as descriptive method provides a mechanism for users and software tools to validate the configuration files. The design of a new user interface facilities the system design and helps to specify and manage configurations through the use of xml files, as a result, workflow is speed up considerably. Using this interface a dual objective is met. First, it has managed to avoid the control engineer necessarily have extensive knowledge of the defined model. Second, it has been possible to obtain a tool able to generate configuration files which meet the specification of the model.

On the other hand, the use of a communication system based in DCPS has facilitated the dissemination of information among the different nodes of the distributed system. The implementation of the system based on the publish/subscribe model has proven to be an efficient form of communication. The structures of these communications have been presented.

Finally, experiments on a physical environment have allowed verifying proper operation of the user interface as well as the validation of the defined model. Moreover, these experiments have allowed to verify that the new settings are applied dynamically.

9 FUTURE WORK

From the conclusions obtained after the development of the user interface, the ability to develop new work where the system is able to monitor the status of each node is proposed. The objective would be to parameterize the information transmitted in the DCPS topics. This solution allows to check the correct system performance, assess its condition and to detect anomalous situations. Likewise, a correct detection may involve the management of secure routines to avoid undesirable

ISBN-13: 978-84-697-0589-6

situations in the controlled system. This is especially important in critical systems. Similarly, a monitoring system allows an alarm management using QoS. If a critical variable does not meet quality of service, the tool could start executing safe routines.

In addition, the monitoring system can be used as debugging tool because it would be possible to observe the communication between components. In this sense, the control engineer can check in real time if your design is meeting the expected functionality.

Moreover, it also proposes to include namespaces within the communications system for the purpose of making a more efficient management. The aim is to make modifications to a group of components rather than modify them individually as at present. The guidelines to create groups of components would be based on its performance, features, etc. On the other hand, the control engineer would be able to monitor the status of these groups of components.

Acknowledgment

This work has been supported by the Spanish Science and Innovation Ministry MICINN under the CICYT project COBAMI: DPI2011-28507-C02- 01/02 and the "Real time distributed control systems" of the Support Program for Research and Development (PAID-06-12) Research Vice-Chancellor of the Polytechnic University of Valencia (SP20120834). The responsibility for the content remains with the authors.

References

[1] Albertos, P., Crespo, A. & Simó, J., (2006) A key concept in embedded control systems. 4th IFAC Symposium on Mechatronic Systems.

[2] Biron, P., Malhotra, A., & World Wide Web Consortium. (2004). XML schema part 2: Datatypes. World Wide Web Consortium Recommendation REC-xmlschema-2-20041028.

[3] CORBA, O., & Specification, I. I. O. P. (1999). Object Management Group.

[4] Coronel, J.O., Blanes, F., Simó, J. & Nicolau, V., (2008) Middleware de kernel de control para el desarrollo de aplicaciones en sistemas empotrados de tiempo real. XXIX Jornadas de Automática.

[5] Crespo. A., Albertos, P., Balbestre, P., Vallés, M., Lluesma, M. & Simó, J., (2006) Schedulability issues in complex embedded control systems. IEEE International Conference on Control Applications.

[6] Fialli, J., & Vajjhala, S. (2003). The Java architecture for XML binding (JAXB). JSR, JCP, January.

[7] Karsai, G., Neema, S., & Sharp, D. Model-driven architecture for embedded software: A synopsis and an example. Science of Computer Programming, 73(1), 26-38. 2008.

[8] Millers, F., Holz, D. & Behnke, S. rxDeveloper: Gui-aided software development in ROS.

[9] Muñoz, M., Munera, E., Blanes, F., Simó, J. & Benet, G., (2013) Event driven middleware for distributed system control. XXXIV Jornadas de Automática.

[10] Object Management Group (OMG), (1995) The Common Object Request Broker (CORBA): Architecture and Specification. Technical report, Object Management Group.

[11] Palopoli, L., Lipari, G., Abeni, L., Di Natale, M., Ancilotti, P., & Conticelli, F., (2001) A tool for simulation and fast prototyping of embedded control systems. In ACM SIGPLAN Notices (Vol. 36, No. 8, pp. 73-81). ACM.

[12] Pot, E., Monceaux, J., Gelin, R., & Maisonnier, B., (2009). Choregraphe: a graphical tool for humanoid robot programming. In Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on (pp. 46-51). IEEE.

[13] Luján, J. L. P., Ten, J. E. S., & Yague, J. L. P. El modelo de comunicaciones DCPS. ISO 690. 2009

[14] Simarro, R., Coronel, J.O., Simó, J. & Blanes, J.F., (2008) Hierarchical and distributed embedded control kernel. In 17th IFAC World Congress.

[15] "Sistemas Distribuidos. Conceptos y Diseño", 3a edición. Pearson Educación, 2001, ISBN: 8478290494

[16] Twin Oaks Computing, Inc (2011) Communications Middleware and DDS.

[17] Vicente Nicolau and Manuel Muñoz and Jose Simó. KertrolBot Platform. SiDiReLi: Distributed System with Limited Resources. Institute of Control Systems and Industrial

ISBN-13: 978-84-697-0589-6

Computing - Polytechnic University of Valencia. Technical report, 2011.

[18] Wain, R. & Ashworth, M, (2005). A Java GUI and Distributed CORBA Client-Server Interface for a Coastal Ocean Model. Council for the Central Laboratory of the Research Councils.

[19] Thompson, H. S., Beech, D., Maloney, M., & Mendelsohn, N. (2004). XML schema part 1: structures second edition. 2004-10. http://www. w3. org/TR/2004/REC-xmlschema-1-20041028.

ISBN-13: 978-84-697-0589-6