# Dynamic Composition and Reconfiguration of Internet-scale Control Systems

Evangelos Pournaras
Faculty of Technology, Policy and Management
Delft University of Technology
2628 BX, Delft, The Netherlands
Email: e.pournaras@tudelft.nl

Mark Yao
and Ron Ambrosio
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA
Email: {markyao,rfa}@us.ibm.com

***Abstract*—The runtime and life-cycle of distributed cyber-physical ecosystems is highly unpredictable and challenging to manage. A large number of spatially located control elements organize distributed control systems. These systems require dynamic interactions of their control elements during their runtime and life-cycle to tolerate failures, balance their computational load, scale and meet application requirements. This paper addresses the problem of dynamic composition and reconfiguration of these dynamic interactions in Internet-scale control systems. Rather than introducing a new middleware or mechanism with a limited applicability and integration in such systems, this paper introduces the 'binding control model' designed as a control system for higher abstraction, integration and applicability in distributed control systems. The binding control model, its possible granularity levels and its applicability are illustrated in this paper.**

## I. Introduction

Distributed control systems and their applications emerge to highly complex, large-scale cyber-physical ecosystems. Application domains of distributed control, such as electrical power systems, transportation systems and water resource management, are built by spatially distributed and interconnected control elements. Such operational environments appear complex dynamics and multiple uncertainties, e.g. network failures, network delays and security attacks. Control elements need to become more autonomous, loosely coupled and self-aware of their cyber-physical ecosystem to self-compose themselves into robust control systems that meet complex application requirements.

This challenge can be met when the interactions and information flow in control systems are reconfigured and composed in a dynamic fashion during their lifetime. Dynamic composition and reconfiguration refers to how a control system of control elements distributed across a network is able to scale in number and evolve during runtime by modifying and self-organizing the binding scheme of its elements. This binding scheme is defined as the logical input and output connections/wirings between these control elements and forms the application graph of a distributed control system.

This paper identifies a gap when control systems and their applications are modeled, designed and deployed in Internet-scale distributed environments. Existing composition and reconfiguration approaches in distributed control systems are out of the scope of a control system itself and left for system composers or external dedicated systems for this purpose [3], [8], [21]. This separation of control systems and their applications from tools and systems to compose them under-emphasizes the importance of dynamic composition and reconfiguration within distributed cyber-physical ecosystems and results in (i) an offline static composition [5] and (ii) hard integration of control applications in these environments [16].

To bridge this gap, the *binding control model* is introduced in this paper for managing the input and output information exchange within distributed control systems. This model is designed as a control system to allow a higher flexibility and integration in the runtime environments of control applications. This model is based on (i) information discovery, (ii) decision-making and finally (iii) reconfiguration of the information exchanged within distributed control systems.

The contributions of this paper are outlined as follows:

- The introduction and design of the binding control model as a control system for the dynamic composition and reconfiguration of information flow in distributed control applications (Section III).
- The introduction of three possible granularity levels for the instantiation of the binding control model in large-scale distributed environments (Section IV).
- The illustration of an actual applicability and integration of the binding control model in the Internet-scale Control System (iCS), a distributed embedded control platform used extensively in critical application domains such as the energy domain [1] (Section V).

The technical aspects and benefits of the binding control model are illustrated in detail in this paper.

## II. Problem Statement

This paper focuses on the problem of dynamic composition and reconfiguration of applications in distributed control systems.

A distributed control system consists of spatially distributed *control elements* that exchange *signals* over a communication network, e.g. the Internet. A control element is an embedded device, a piece of software or a cyber-physical system that has the role of a *sensor*, a *controller* or an *actuator*. A sensor monitors a device, or another piece of software and generates output signals based on sensing information. Controllers process input and generate output signals based on an application algorithm. Actuators receive input signals and undertake actions, e.g. turning on or off the thermostat of a device. In a distributed control system, these elements can be designed to remotely exchange signals and coordinate their objectives to enable distributed control applications in a wide range of cyber-physical domains: electrical power systems, transportation systems and water resource management.

This paper refers to the composition and configuration of distributed control applications as the logical software bindings (wirings) of input and output (I/O) information signals between spatially distributed software control elements. More specifically, the *composition* refers to the design of the application graph to enable a complex application logic, that is the selection of I/O bindings between control elements. The *configuration* is the supporting mechanism to instantiate and manage these bindings. Note that this paper specifically focuses on the composition of the logical bindings between control elements and is distinguished from the composition of the actual elements addressed in related work [2].

Most of the existing applications of distributed control are composed before their actual deployment and runtime. Their configuration is usually part of their initialization [5] and not their runtime. This can degrade their robustness tremendously. Uncertainties of distributed systems such as failures, delays and security attacks can interrupt the I/O signals between control elements of an application. Dynamic composition and reconfiguration organizes the bindings between control elements on-the-fly by removing, adding and moving bindings among online control elements. Dynamic binding can provide a wide range of new possibilities for a distributed control application to evolve as illustrated in the following examples:

- Fault-tolerance by routing I/Os to alternative and available control elements in case some of them are disconnected.
- Information discovery by updating the I/Os bindings to capture new information from different control elements.
- Load-balancing by rebinding input signals from an overloaded control element to an underloaded one.
- Integration and scalability by adding and connecting new control elements to an existing cluster of control elements.

Existing approaches [6], [8], [16] are usually centralized based on single management components or administration tools. Such approaches are not scalable and require a large amount of monitoring and state information that is not always available among enterprise environments. In some cases [21], solutions are realized out of the scope of control systems based on agent-based paradigms. This paper proposes a binding control model for dynamic composition and reconfiguration of applications in distributed control systems. The modeling of dynamic composition and reconfiguration as an actual control system is the design challenge that this paper tackles.

## III. The Binding Control Model

This paper introduces the *binding control model* for the composition and reconfiguration of I/O bindings in distributed control systems. Note that, this model is designed and can be instantiated as a control system. Therefore, it allows a higher level of abstraction, integration and applicability in the area of distributed control systems. Figure 1 illustrates the introduced binding control model.

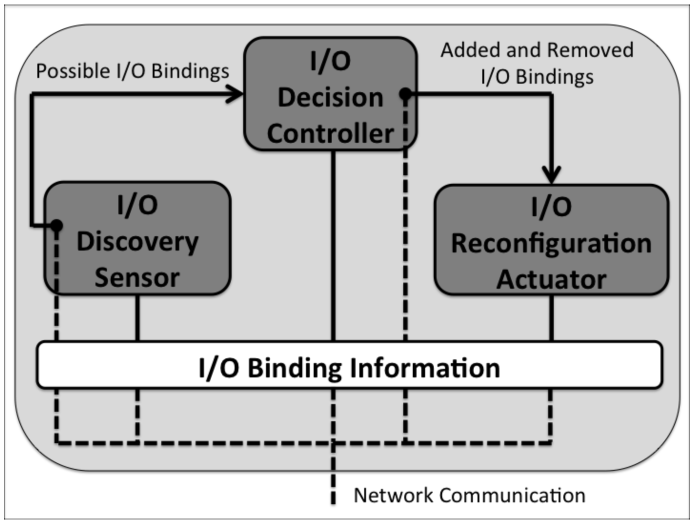

Figure 1. The binding control model for managing I/O bindings in distributed control systems

This model assumes a communication network and a system for storing and managing the I/O binding information of an application. Note that an application is a group of control elements whose I/O bindings are required to be dynamically composed and reconfigured. The communication network supports remote calls and communication protocols serving application control elements as depicted by the dashed lines of Figure 1. It is the same network communication systems on which the binding control model is based.

An I/O binding between two control elements of an application can be represented as two triples of information: (i) the input or output identifier respectively of a control element, (ii) the respective control element identifier and (iii) the identifier of the physical machine, e.g. the IP address and port number, that hosts the respective control element. The I/O binding information system consists of this type of information stored and managed locally within each instance of the software platform running application control elements. Such a software platform can be a virtual machine or a distributed middleware. An I/O binding information system is based on different communication models [10]. In a *peer-to-peer* model, I/O bindings correspond to an one-to-one communication between control elements. Alternatively, a *publish-subscribe* model allows a more loosely coupled binding between control elements. The communication network is used as an event bus in which control elements can publish (output) and subscribe (input) to certain events. Note that, both models can be adopted for higher flexibility [17], [20]. Section V-B shows an alternative way to engage both models based on the binding control model.

The binding control model consists of three control elements: (i) the *I/O discovery sensor*, (ii) the *I/O decision controller* and (iii) the *I/O reconfiguration actuator*. These three control elements are binded to each other and are coupled with the I/O binding information system as depicted by the solid lines of Figure 1. The control elements of the binding control model use the I/O binding information system to manage (i) the bindings of the application control elements and (ii) the bindings between each other. They act as a control system sensing and actuating on the I/O binding information of an application. Therefore, the binding control model can be instantiated in the same software runtime environment on which the control elements of an application run.

## *A. The I/O Discovery Sensor*

Dynamic composition and reconfiguration requires the discovery of new information concerning the possible I/Os to which a control element binds. An input or output of a control element is a *possible* bind to an output or input of another control element respectively if the types of their information match. For example an integer output matches with an integer input but a float output does not match with a string input. Changes in the I/O types are assumed not to occur during the runtime of a distributed control system. However, control element may fail and therefore certain I/O types may extinct. New control elements with new I/O types may be added during runtime. These more dynamic scenarios are part of future work.

The actual information discovered by this sensor is the *I/O descriptors*. A descriptor is a set of information containing the I/O triple, that is the identifiers of the I/O, control element and host. The descriptor may also optionally contain other types of information related to an application. For example, a descriptor of an energy provider controller may contain QoS information related with a type of energy source. A descriptor with the consumption profile of an end-user device controller may be used for demand-side management. The information within the I/O descriptor is critical for the selection and establishment of I/O bindings. The I/O decision controller handles this information and its role is discussed in Section III-B.

The actual instantiation of the I/O discovery sensor is a design and implementation issue related with (i) the available resources of the network communication system (ii) the application and (iii) the software platform on which the application and the binding control model are developed and deployed. Different discovery mechanisms and protocols can be engaged, such as flooding [14], random walks [7], gossiping [13] and DHT overlays [15]. Section V-B illustrates an instantiation of the I/O discovery sensor based on gossiping.

## *B. The I/O Decision Controller*

The I/O decision controller selects a number of I/Os to bind with or unbind from. A list of possible I/O bindings (descriptors) is provided by the I/O discovery sensor. This is the input of the I/O decision controller. Based on its decision-making scheme discussed below, the selected descriptors are provided to the I/O reconfiguration controller.

Decision-making can be based on (i) an administration scheme or it can be (ii) autonomous and part of the application design. Both schemes can be adopted as well under different situations.

In the first case, the application that requires dynamic binding is highly critical. Continuous and real time monitoring is enforced and any failure to meet application requirements may have catastrophic effects. For example, existing air-controlling system cannot operate in a fully automated fashion. The uncertainties of weather, physical environment and mechanical failures in aircrafts require highly complex control systems that have not yet been transformed to fully automated ones. Similar issues concern other application domains that remain, to a certain degree, controlled by system operators and administrators. The I/O decision controller allows administrators to interact with a binding control system. Rules, commands, or policies together with constraints and other collected information build a knowledge representation of the decision-making scheme. This knowledge representation depends on the application design and requirements but also on the supported platform that runs the application. Section V-C illustrates a scenario of how an administrator uses an I/O decision controller to manage dynamically bindings of application control elements. Note that, in the case of an administration scheme, the I/O discovery sensor is a complementary element supporting this scheme. The information of the I/O discovery sensor can be provided, in theory, by an administrator if it is available.

In the second case, decisions are automated and are based on a scheme designed to serve a specific type or class of applications. Multi-agent systems, genetic algorithms, self-organization algorithms and evolutionary optimization are some examples of such design schemes with autonomous decisions [12], [22]. A common characteristic in these approaches is the existence of a *fitness function* that can support the I/O decision controller in the binding control model. A fitness function receives as input the I/O descriptors and other local information related to an application. This function ranks or sorts the possible I/O bindings according to a criterion, their fitness, as defined by the application. For example, assume that the criterion for I/O binding selections is the bandwidth of the nodes over which the control elements are deployed. The euclidean distance is a simple evaluation metric to compare the fitness of I/O bindings. Other or additional criteria related to the application requirements can be engaged and blended [19] in the fitness function. Note that, generic fitness functions that satisfy different applications are challenging to design and are part of ongoing research [18].

## *C. The I/O Reconfiguration Actuator*

The I/O reconfiguration actuator is the actual element that adds or removes I/O bindings to or from the control elements of an application. Actuation concerns the local and remote calls to the I/O binding information system of the control elements that are binded or unbinded. The I/O decision controller provides all the required information facilitated in the I/O descriptors for reconfiguring the I/O bindings: the identifiers of the I/Os, control elements and hosts.

In a distributed control system based on a peer-to-peer communication model, direct connections are established or dropped between the control elements of an application. If the control elements are remotely located, the I/O reconfiguration actuator performs remote calls to realize the binding and change the information in the remote I/O binding information system. In a peer-to-peer communication model, both control elements are affected when a bind is added or removed.

In a publish-subscribe model adopted for communication, control elements of an application are loosely coupled. I/O bindings are added and removed by subscribing and unsubscribing to certain output events. Therefore, reconfiguration is required only in the location of control elements that bind their input. However, the lower lever routing of published events may also require changes. For example, published events should not be routed to locations in which there are not subscribers.

## IV. Model Granularity

The binding control model can be instantiated at different granularities. In this paper, granularity refers to the level on which the binding control model is designed, implemented and deployed within the environment of a distributed control system and its applications. Understanding the different possible granularity levels is crucial for the general applicability of this model in decentralized control systems.

This paper discusses three possible granularity levels at which the binding control model can be instantiated:

- **System-level**: The binding control system acts as an external system or service and may possibly have its own runtime environment. In this case, its design and implementation is not necessarily related or integrated with the software design and implementation of the distributed control application. From the viewpoint of the software platform that runs the application, the binding control system is a centralized service. However, the actual binding control system may be transparently distributed and deployed in different physical machines as depicted in Figure 2a.
- **Node-level**: The binding control system is part of the software platform and runtime environment that also runs the distributed control application. This software platform is installed in multiple nodes with the binding control model being part of the runtime. In this case, a local binding control system in every node is responsible for the dynamic binding of the control elements that run in this specific node. This means that the I/O binding information system in every node is sensed and actuated by an I/O discovery sensor, an I/O decision controller and an I/O reconfiguration actuator. Therefore, a number of remotely located binding control systems in different nodes coordinate their actions to discover, select and finally reconfigure the I/O bindings of their local control elements. Figure 2b illustrates the node-level instantiation of the binding control model.
- **Element-level**: The binding control system is designed and developed to serve a specific control element of an application. In this case, there is a one-to-three elements relationship, that is, a correspondence of an application control element to an I/O discovery sensor, an I/O decision controller and an I/O reconfiguration actuator. This approach is highly decentralized and provides a higher flexibility and autonomy for control elements to self-compose and self-reconfigure their bindings. Figure 2c illustrates the concept of an element-level instantiation of the binding control model.

Note that in the node-level and element-level instantiations of Figure 2, the binding control system appears locally located with the application control elements whose I/O bindings manages. This is not necessarily enforced as the binding control system is also a control system application and it can be deployed in separate allocated nodes for the purpose of dynamic binding composition and reconfiguration.

The selection of the most relevant and appropriate granularity level for the model instantiation is a design issue and depends on multiple factors. The flexibility of the runtime environment, the API, the type, complexity and requirements of the application, the available computational and network resources and the security enforced in each granularity level by the runtime environment are only some of these factors to be considered.

## V. Model Applicability

This section illustrates the applicability of the binding control model in cyber-physical embedded systems and specifically in the Internet-scale Control System (iCS) [1]. iCS is a Java Micro Edition (JME) lightweight runtime environment for distributed control applications. iCS has been used in various domains of large-scale control application such as energy demand-side management in the Olympic Peninsula GridWise Demonstration Project [9] and is currently the core communication technology used in the Pacific Northwest Smart Grid Demonstration Project[1].

iCS is based on loosely coupled control elements that are binded based on the publish-subscribe communication model. iCS control elements are distributed among iCS nodes that provide the runtime environment. An iCS node has a local publish-subscribe system and two proxies for remotely publishing and subscribing I/Os. Synchronous and asynchronous communication is supported by allowing control elements of an application and system administrators to issue management commands and exchange messages. The actual network communication is realized by different transport mechanisms providing different levels of decentralization and security. Currently, iCS supports (i) a TCP peer-to-peer communication,

[1] www.pnwsmartgrid.org

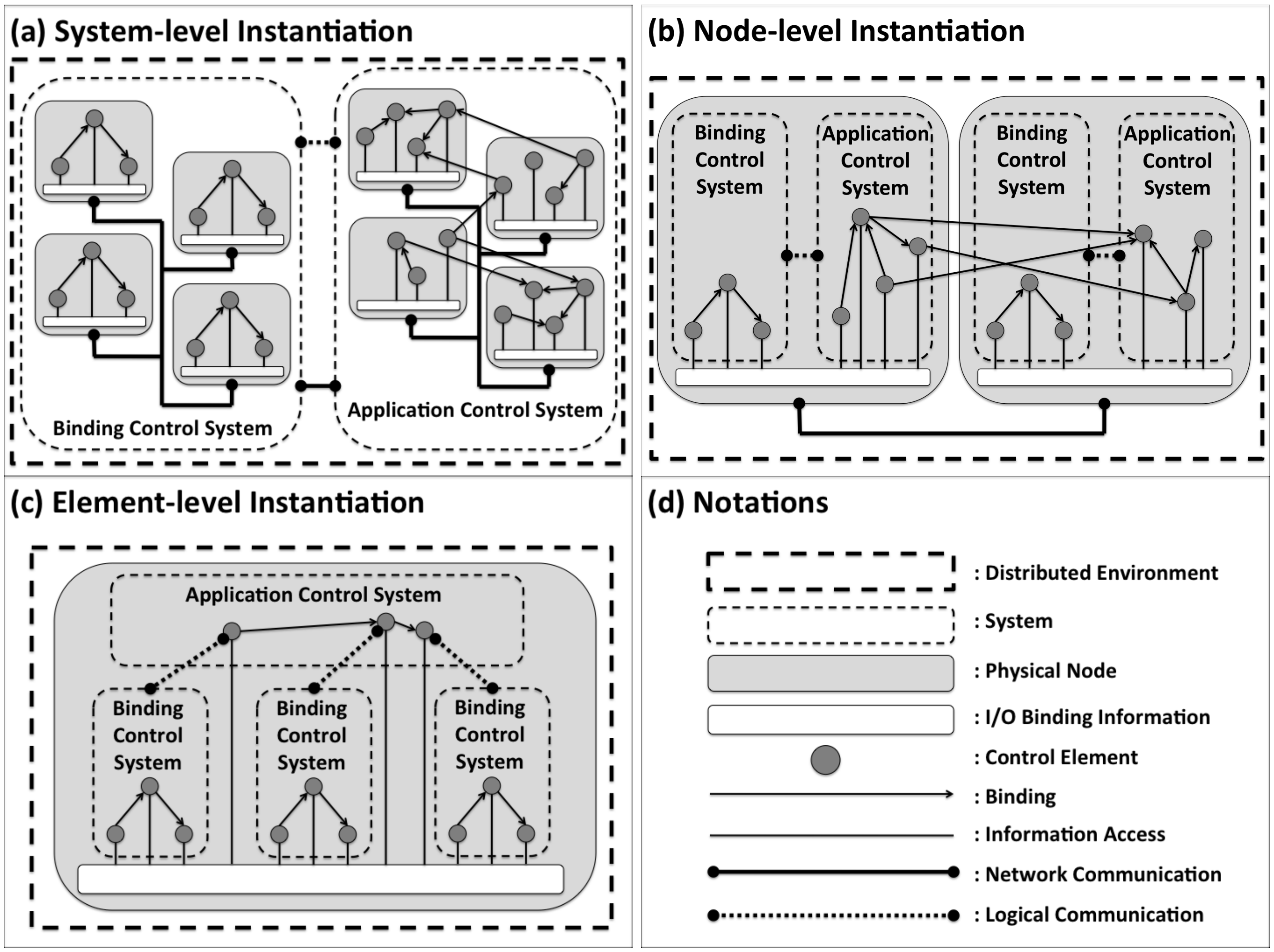

Figure 2. Three possible granularity levels for the instantiation of the binding control model.

(ii) the MQtt broker system [11], (iii) RMI and (iv) the Harmony overlay network [4]. Figure 3a illustrates a simplified view of the iCS architecture.

An I/O binding between two control elements locally located in an iCS node requires one and only one topic subscription of the target (input) control element in the local publish-subscribe system. In contrast, an I/O binding between two remotely located control elements requires two subscriptions. Figure 3b illustrates a simple scenario of an underlying remote peer-to-peer communication corresponding to a logical binding between two remote control elements. One topic subscription is performed by the input control element as in the case of a local I/O binding. A second subscription of the same topic is required by the remote subscriber proxy in the publisher's side. When this proxy receives an event from the publisher of the topic to which it is subscribed, it forwards the event to the remote publisher proxy in the other iCS node where the subscriber is located using the network adapters. In this way, iCS maps the logical bindings between control elements to an actual network communication and routing.

The composition of a control application is a series of subscriptions performed during an initialization phase. The iCS runtime reads two types of files, (i) the binding map and (ii) the network map. The former contains information about the I/O bindings, that is which I/Os of control elements are binded. The latter contains information about the physical location of the iCS nodes and the distribution of control elements in these nodes. This composition approach is static and does not allow any reconfiguration and evolution during runtime as discussed in Section II.

This section illustrates the instantiation and integration of the binding control model in iCS. The node-level granularity is chosen for the model instantiation over the element-level granularity for the reason of reducing the computational and memory resources in iCS nodes. Introducing fewer control elements reduces the cache allocation and the number of running threads in iCS. However, an actual performance comparison of the three granularity levels in iCS is part of future work.

## A. Reconfigurations

Reconfigurations are the first step for the instantiation of the binding control model. This means that the introduced

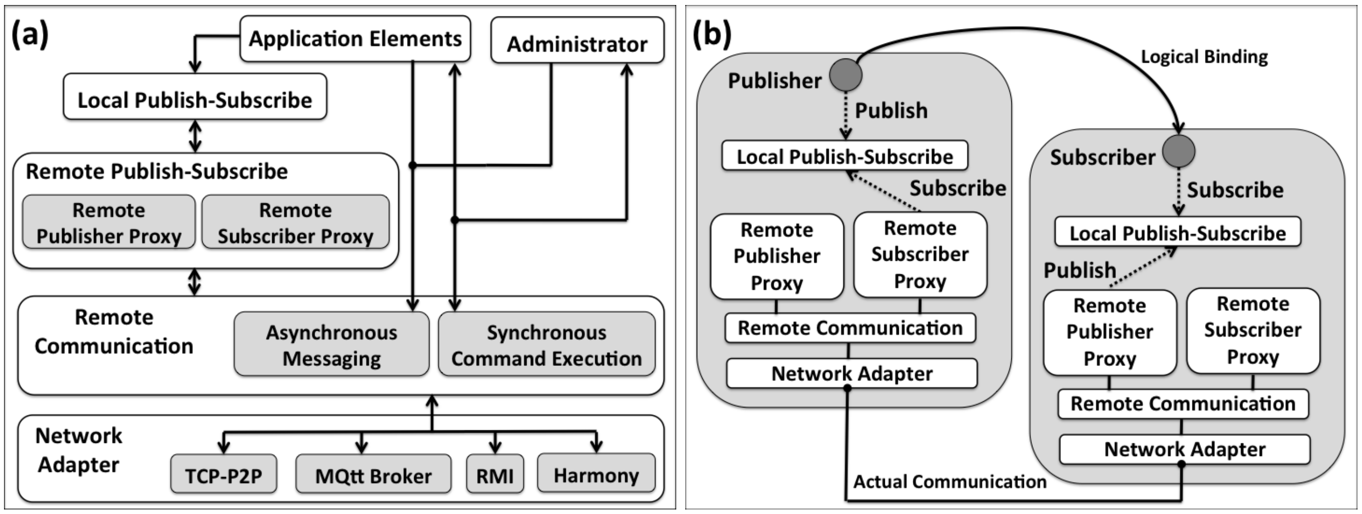

Figure 3. The Internet-scale Control System. (a) The local architecture of an iCS node. (b) Remote communication between two remotely binded control elements.

binding control system should be coupled appropriately with the underlying mechanism that manages the I/O binding information system, in this case, with the publish-subscribe system. The synchronous command execution mechanism of iCS is used for this purpose. The I/O reconfiguration actuator, as an application control element, issues a *'bind'* command that, in case of iCS, has the form of *bind -/+* {*source descriptor*} {*target descriptor*}. The '-' indicates a removal and the '+' an addition of a binding between the source (output) and target (input) control elements. The I/O information about these control elements is contained in the descriptors.

The 'bind' command, as any other iCS command, can be executed remotely in different iCS nodes. Therefore, when this command is issued, the iCS runtime forwards the command, transparently from the control elements, to the affected iCS nodes and waits for the execution result. In case of two remote control elements, the 'bind' command is successful if and only if the I/Os types match and both subscriptions (or unsubscriptions) illustrated in Figure 3b are performed successfully.

The use of the iCS command execution is the only requirement for the management of the publish-subscribe system by the I/O reconfiguration actuator. Therefore, this integration approach does not require any changes in the publish-subscribe system, the remote communication system or the network adapters.

## *B. Information Discovery*

Binding reconfigurations require knowledge about the descriptors of the control elements. iCS does not support an explicit lookup mechanism for making this information available. Therefore, a gossiping discovery mechanism is introduced in iCS to support the I/O discovery sensor. Gossiping is a self-organization mechanism that provides a high binding connectedness between control elements, preventing the clustering of the application graph in case of failures in iCS nodes or iCS control elements. The core idea of gossiping is that a number of *gossiping controllers* store and update periodically their *views* of other gossiping controllers. A gossiping controller is a control element and its view is a partial list of descriptors in the system.

The view of the gossiping controller is sensed by the I/O discovery sensor for new descriptors of control elements. The updates in the view and the exact gossiping algorithm description are based on the peer sampling service [13]. Note that this paper does not focus on the illustration of the convergence and performance of gossiping protocols but rather on the integration of gossiping in iCS as a supporting mechanism of the I/O discovery sensor. Readers are referred to related work for quantitative results about gossiping [13], [18].

The principle of gossiping is the continuous push-pull exchange of views between different gossiping controllers. Therefore, gossiping controllers change their binding connections continuously and exchange their views in a peer-to-peer fashion. The main challenge with the integration of gossiping in iCS is how to enable a peer-to-peer communication by using the publish-subscribe system that manages the I/O binding information.

The gossiping mechanism of iCS takes advantage of the reconfigurations performed by the binding control system. Algorithm 1 and 2 illustrate the active and passive threads respectively of the gossiping controllers. Every time that a gossiping controller periodically initiates a push-pull gossip information exchange (line 1, Algorithm 1), it creates a new binding (line 4, Algorithm 1). The input and output type of this binding is the *gossip* that contains three types of information (line 6, Algorithm 1): (i) a 'push' or 'pull' flag to distinguish the inputs received during a gossip exchange (line 3 and 10, Algorithm 2), (ii) the descriptor of the source gossiping elements that initiates the 'push' or 'pull' output (line 6, Algorithm 2) and (iii) the view of the source gossiping controller used by the target gossiping controller to update its local view (line 13, Algorithm 2). When the two gossiping controllers are binded, the gossiping initiator can publish its gossip (line 7, Algorithm 1). The subscribed gossiping controller receives the gossip (line 2, Algorithm 2) and repeats the same process. Finally, when the gossip exchange is complete, the bindings between the two gossip controllers are removed (lines 11 and 12, Algorithm 2).

Note that, the selection of gossiping controllers (line 3, Algorithm 1) and the view update (line 13, Algorithm 2) are part of the peer sampling service and they do not influence the integration of gossiping in iCS as a supporting mechanism of the I/O discovery sensor. These operations are illustrated in detail in related work [13].

### C. Decision-making

An I/O discovery sensor senses the gossiping controller for new I/O descriptors of application control elements and provides (outputs) these descriptors to the I/O decision controller. A simple decision-making scenario is considered in this paper. An administrator, system operator or application composer is interested in updating the application graph, that is the binding between the control elements, without any knowledge about the actual deployment in the underlying network topology. In other words, the administrator has only knowledge about the new binding map of an iCS application and not about the network map. This binding map is provided as an input in the I/O decision controller.

The possible descriptors that are published to the I/O decision controller by the I/O discovery sensor contain the required network information about the physical location of each control element. Therefore, this information is not required to be provided by, for example, an application composer as it is sensed by the I/O discovery sensor and is made available to the I/O decision controller. A control element in a binding map corresponds to a descriptor of this control element disseminated by the gossiping controllers and discovered by the I/O discovery sensor. In this way, a local matching check can be performed by the I/O decision controller: (i) Assume the set of the existing I/O bindings that are not defined in the new binding map. Then, the possible descriptors received by the I/O decision controller that match this subset are sent to the I/O reconfiguration actuator to perform a binding removal. In contrast, (ii) assume the set of bindings defined in the new binding map that are not currently established. Then, the possible descriptors received by the I/O decision controller that match this subset are sent to the I/O reconfiguration actuator to perform a binding addition.

The introduction of more complex decision-making schemes in the I/O decision controller are part of on-going work. Self-composition scenarios are specifically considered in which decisions are driven by the application control elements.

## VI. Conclusions and Future Work

This paper addresses the problem of dynamic composition and reconfiguration in large-scale decentralized control systems. Rather than introducing a new middleware service or other dedicated mechanisms for solving this problem, this paper provides a generic modeling and design solution that benefits from a higher level of abstraction, integration and applicability in the area of distributed control systems. The binding control model is designed as a control system and is able to be instantiated at different granularity levels. This flexibility serves the higher applicability of this model in various embedded control platforms and applications. This paper actually shows such an applicability in the Internet-scale Control System (iCS), a distributed embedded control platform used extensively in critical demonstration projects of electrical power management [1].

A quantitative evaluation of the benefits of the binding control model in such demonstration projects and other application domains is part of ongoing and future work. Performance analysis is required to clarify and actually compare the relevance of each granularity level in different application scenarios.

## Acknowledgment

The authors would like to thank Martijn Warnier, Frances M.T. Brazier, Jeff Kephart and Brian Gaucher for their help to make this collaboration possible.

## References

[1] R. Ambrosio, A. Morrow, and N. Noecker. e-business control systems. In *Proceedings of the 2nd International Conference on Computing, Communications, and Control Technologies*, pages 91–96, University of Texas, Austin, TX, 2004. IEEE Computer Society.

[2] C. Angelov and K. Sierszecki. A Component-Based Framework for Distributed Control Systems. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pages 20–27. IEEE.

[3] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. Dynamic binding in mobile applications. *Internet Computing, IEEE*, 7(2):34 – 42, 2003.

[4] P. Dube, N. Halim, K. Karenos, M. Kim, Z. Liu, S. Parthasarathy, D. Pendarakis, and H. Yang. Harmony: holistic messaging middleware for event-driven systems. *IBM Syst. J.*, 47:281–287, April 2008.

[5] T. Genß ler and C. Zeidler. Rule-Driven Component Composition for Embedded Systems. In *Intl. Conf. on Software Engineering (ICSE): Workshop on ComponentBased Software Engineering*, 2001.

[6] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems - WOSS '02*, page 33, New York, New York, USA, Nov. 2002. ACM Press.

[7] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Performance Evaluation*, 63(3):241 – 263, 2006. P2P Computing Systems.

[8] M. Guler, S. Clements, N. Kejriwal, L. Wills, B. Heck, and G. Vachtsevanos. Rapid Prototyping of Transition Management Code for Reconfigurable Control Systems. In *Proceedings of the 13th IEEE International Workshop on Rapid System Prototyping (RSP'02)*, RSP '02, pages 76–, Washington, 2002. IEEE Computer Society.

[9] D. Hammerstrom, T. Oliver, R. Melton, and R. Ambrosio. Standardization of a hierarchical transactive control system. In *Proceedings of the Grid Interop '09 Conference*, 2009.

[10] B. Heck, L. Wills, and G. Vachtsevanos. Software technology for implementing reusable, distributed control systems. *IEEE Control Systems Magazine*, 23(1):21–35, Feb. 2003.

[11] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. Mqtt-s - a publish/subscribe protocol for wireless sensor networks. In *COMSWARE*, pages 791–798, 2008.

[12] M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321 – 2339, 2009. Gossiping in Distributed Systems.

**Algorithm 1** The active thread of the gossip controller.

```
1. do every T period
2.      sourceDescriptor=myDescriptor;
3.      targetDescriptor=myView.select();
4.      bind + sourceDescriptor targetDescriptor;
5.      sourceView=myView;
6.      sourceGossip={'push',sourceDescriptor,sourceView};
7.      publish(sourceGossip);
```

**Algorithm 2** The passive thread of the gossiping controller.

```
1. do for ever
2.      sourceGossip=publishSubscribeSystem.receive();
3.      if sourceGossip is 'push'
4.      then sourceDescriptor=myDescriptor;
5.           sourceView=myView;
6.           targetDescriptor=sourceGossip.sourceDescriptor;
7.           targetGossip={'pull',sourceDescriptor,sourceView};
8.           bind + sourceDescriptor targetDescriptor;
9.           publish(targetGossip);
10.     if sourceGossip is 'pull'
11.     then bind - sourceDescriptor targetDescriptor;
12.          bind - targetDescriptor sourceDescriptor;
13.     myView.update(sourceGossip.sourceView);
```

[13] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):8–es, Aug. 2007.

[14] S. Jiang, L. Guo, and X. Zhang. Lightflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. In *ICPP*, pages 627–635, 2003.

[15] Y.-J. Joung, C.-T. Fang, and L.-W. Yang. Keyword search in dht-based peer-to-peer networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, ICDCS '05, pages 339–348, Washington, DC, USA, 2005. IEEE Computer Society.

[16] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. In *Future of Software Engineering (FOSE '07)*, pages 259–268. IEEE, May 2007.

[17] P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, pages 1–8, New York, NY, USA, 2003. ACM.

[18] E. Pournaras, M. Warnier, and F. M. T. Brazier. Adaptation strategies for self-management of tree overlay networks. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, pages 401–409, 2010.

[19] G. Tan. Performance Analysis and Improvement of Overlay Construction for Peer-to-Peer Live Streaming. *Simulation*, 82(2):93–106, 2006.

[20] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, pages 1–8, New York, NY, USA, 2003. ACM.

[21] L. Wang, S. Balasubramanian, and D. H. Norrie. Agent-based Intelligent Control System Design For Real-time Distributed Manufacturing Environments. pages 115–152. In Working Notes of the Agent Based Manufacturing Workshop, 1998.

[22] W. Zhong, J. Liu, M. Xue, and L. Jiao. A multiagent genetic algorithm for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(2):1128–1141, 2004.