Conception of a Type-based Pub/Sub Mechanism with Hierarchical Channels for a Dynamic Adaptive Component Model

Mohamad Ibrahim, Karina Rehfeldt, Andreas Rausch

Technische Universität Clausthal
38678 Clausthal-Zellerfeld, Germany
email: {mohamad.ibrahim, karina.rehfeldt, andreas.rausch}@tu-clausthal.de

***Abstract*—When attacking the problem of live information dissemination, then publish/subscribe technology plays a key role in crafting an efficient solution. Especially in safety-critical domains like automotive embedded systems a key factor for an efficient publish/subscribe mechanisms is type-safety. We introduce a solution for type-based and semantic publish/subscribe which allows to create hierarchical channels tailored to the needs of an embedded system with physical entities communicating. Our concept builds up on our dynamic adaptive middleware called Dynamic Adaptive System Infrastructure (DAiSI), which allows component configuration at runtime. As a technical medium, we use the industrial standard Extensible Lightweight Asynchronous Protocol (Exlap). Regardless of the fact that our implementation and example pertain to DAiSI and Exlap, our concept is introduced in an integrated framework, which allows the reusability of this model in other application domains.**

Keywords–Component Model; Publish/Subscribe; Channels; Dynamic Adaptive Systems; Embedded Industrial Systems.

I. INTRODUCTION

Many application domains require data dissemination, like stock market data updates, online advertising, asynchronous events in graphical user interface (GUI) and many others [1]. Our focus lies on signals and live data dissemination in embedded and industrial systems. This domain implies certain requirements and restrictions the concept has to commit to. These environments require strict distinctions between data without the possibility of mistaking one for another (type safety). Also, the performance is critical in industrial environments where compared to content-based, ontology or internet wide pub/sub systems [2], [3] scalability and usability plays a more crucial role due to resource limitations. Type-based pub/sub also gives us several advantages that suit the embedded industrial systems environment like encapsulation, application-defined events, open content filters [4], and event semantics which we exploit here to provide the physical diversity of the same type or component.

This work is not intended to address the problem of providing a uniform interface for heterogeneous information sources, but to provide a light-weight system that satisfies specific industrial needs and yet flexible enough for wide-range of applications in the domain of Internet of Things. Examples of such domains can be equipping a car, a house, a factory or any entity of many components with a scheme that can ease information discovery and access from inside and outside the entity. Imagine for example a car, where you have a variety of distance sensors observing the distances around the car. One or more observe the front, some the back and so on. Our goal was to find a concept that presents to the subscriber a scheme that matches a physical entity and describes its components functional relationships, so that the subscriber would seamlessly subscribe to a group of components that share a common function or purpose regardless of their number, position or semantics. In other words, a subscriber should be able to subscribe to all distance sensors at once. But also, the subscriber should be able to subscribe to a specific sub-group or individual, like all distance sensors in the front.

The intention here is to discuss a flavour of pub/sub that is type-based, hierarchical and introduces a new dimension that allows extensibility, so we can represent many physical entities with the same type but different semantics. Figure 1 shows an example for a domain scheme. Engine represents the type (*IEngine*) with the data *speed*. Hierarchical structured, we have cylinders (*ICyclinder*) and a cooling system (*ICooling*). Notice, the domain scheme only states the structure of the data types for publishers, but does not state anything about instances of these publishers. We may have front and rear engine in a specific car domain. So if the domain scheme represents the class in object-oriented programming, then the object counterpart is the physical entity publishing. This arrangement creates two trees, the first is the types hierarchical scheme of the entity with its embedded components. The second represents the real components which has two dimensions: functional dimension which is depicted by the type, like *IEnginge*, and physical dimension which is depicted by the physical semantic, like front or rear engine.

In Section II we introduce some related works in the filed of publish/subscribe. The discussion of our concept begins with introducing the architectural model and system layers in Section III. Section IV focus on the software architecture of the proposed system with elements and APIs explained. We conclude with a disucssion of the proposed solution in Section V.

II. RELATED WORK

The main aim of this work is to introduce a pub/sub system for specific problem yet it can be applied in any other situation, that is why introducing it in an architecture or software stack is important.

Klus et al. [5] - which is the main related and direct previous work - introduce a component model and middleware for dynamic adaptive system that can adapt to different situations by supporting dynamic changing of pre-configured

 ISBN: 978-1-61208-610-1

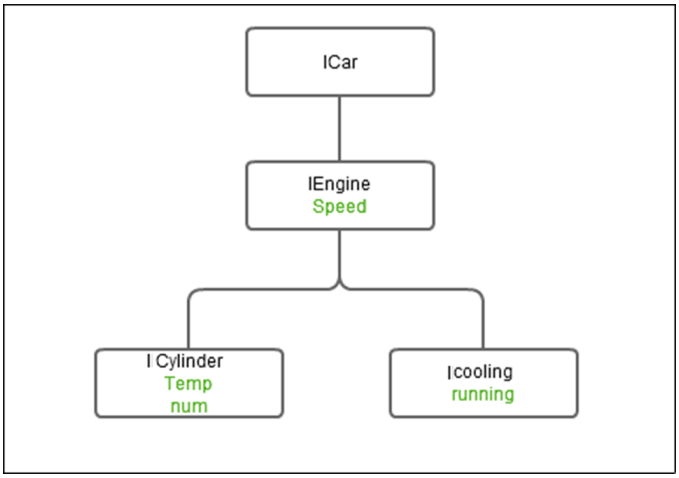

Figure 1. The ICar domain scheme

configurations. However, the proposed model, called DAiSI, is not able to support neither publish/subscribe communication paradigm nor asynchronous data exchanges. That deficiency led to our work that proposes a concept for pub/sub.

Eugster et al. [6] which sets the standards for pub/sub system solutions and gives the fundamentals and several aspects of designing such a system. Liu and Bale give an overview about general pub/sub system ins [7]. Another comprehensive survey of pub/sub systems is given by Filho and Redmiles in [8]. They define several dimensions of pub/sub system and add to them the versatility dimension which means for the concept to be able to adapt to different application requirements. However, what Baldoni et al. [9] are suggesting, is a general concrete system and introducing it in a specific architecture with giving alternatives to different types of applications. We are going to embody our system in this architectural model and make several justified design decisions.

The second category of papers we looked into, are concrete pub/sub systems that have been introduced to solve communication problems in several domains. The first is the semantic Toronto pub/sub system [10], [11], that discusses security on a domain-based infrastructure. Morales et al. [12] introduce a solution for scientific workflow management systems to monitor working processes using a pub/sub system. Bickson et al. [13] introduce a system that utilizes the IP unicast and multicast capabilities to save network resources. Demers et al. [14] take the content-based systems expressiveness to an advanced level depending on finite state automata to express subscription patterns. The given examples show the variety of pub/sub application domains and their different key concepts.

The third category are the type-based pub/sub systems [4], [15], [16], which all reach a consensus that the most prominent features of type-based pub/sub systems are: type-safety, encapsulation, application-defined events, open content filters and event semantics. Since our industrial use-case asks for those features, we decided to make use of type-based pub/sub in our concept.

III. System Overview

There have been a lot of architectural models that specify how to build pub/sub systems. One of the most famous is what Baldoni et al. [9] suggested. Their architecture is structured in the layers Network Protocol, Overlay Infrastructure, Event Routing and Matching. Those layers represent the logical functionality of the components of the pub/sub system. In the following, we will show how our concept is settled in these layers.

A. Network Protocol

The network protocols can vary depending on the environment of the deployment and the application. This layer connects the actual hardware infrastructure of the system [9]. Our realization uses TCP/IP conjugated with Exlap (Extensible Lightweight Asynchronous Protocol) [17]. The upper layers of the system should address all the shortcomings of this layer like security, reliability and unregulated delays. Those subjects are not in the scope of this paper.

Exlap on TCP/IP constitutes the lower layer in the implementation. It is a protocol that follows the client-server paradigm and provides basic interface for pub/sub communications between the client and the server identified by address, port and application-level ID that uniquely identifies any Exlap service. In addition, Exlap provides a discovery service that scans the network for public services. With Exlap, we have an interface for our system components to communicate over the physical layer.

B. Overlay Infrastructure

This layer addresses the organization of the components or nodes, the role of each node and the overall functionality on which the routing of the events rely on [9]. Here comes the role of DAiSI infrastructure, which we use to apply the broker pattern, thus components can act as subscriber, publisher or broker.

First, we take a quick overview on DAiSI before showing how DAiSI fits into the big picture. DAiSI is a dynamic adaptive system infrastructure that lets you create components which can offer certain kinds of services and uses other services in an environment where they dynamically activate the configuration that is of the highest priority or that best matches the existing required services.

For example, a cooling system in a car can be a component that needs to consistently read the temperature from the engine component, and it can provide the status of the cooling system to the car monitor to show it to the driver. This component can hold two configurations in order to dynamically adapt to what is provided. The first configuration can be to provide the status of the cooling system to the monitor, and the second can be to stream the temperature in real time from the engine, applying the logic and provide the status as a service to other components including the monitor, Figure 2 shows the component.

When there is an engine service available, *Conf 2* will be activated, otherwise *Conf 1* is activated. *Conf 1* needs no service in order to be resolved and then provide the *ICooling* service, but the *Conf 2* needs *IEngine* service in order to be resolved and then provide its *ICooling* service.

We employ this DAiSI component representation to provide the required roles in the pub/sub system. Where the broker is the service provided by the system that the publishers and subscribers will connect to. And the required services will take

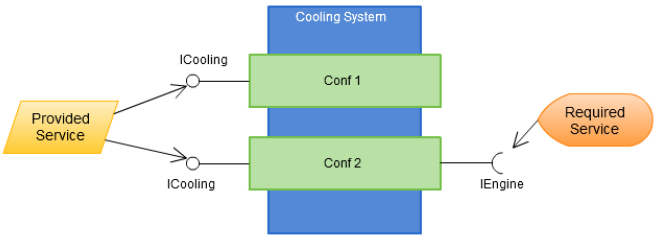

Figure 2. DAiSI Component

the role of the publishers and subscribers of a certain event type. We see in Figure 3 how the above Figure changed in the new context.

Now *Conf 2* will be activated not by the existence of *IEngine* service but when there is a broker service and this broker has *IEngine* being published to it. It is the same as before, but the old relationship is being mediated by the broker.

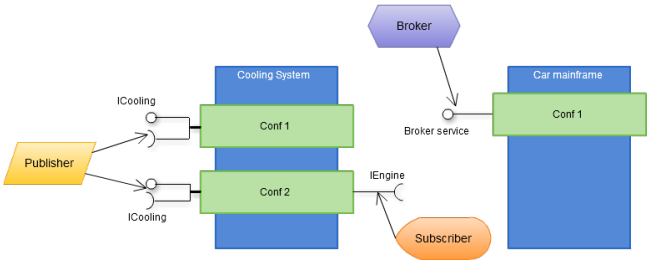

Figure 3. Pub/Sub components

C. Event Routing

In this layer lays the concept of domains. We divide the system space into domains, where every domain has a tree-structured types represent the events available in this domain (see Figure 1. This scheme represents the functional dimension we talked about earlier, and the head of the domain represent the type of the domain which can be conjugated with an ID to depict the physical dimension of the domain and uniquely distinguish the domains of the same type in the system space.

Now assigning an ID to the head of the domain - represented by broker(s) - makes the domain unique in the space with similar domains of the same type.

We will distinguish between two kinds of communications or event routing; inter-domain communication which is done on DAiSI overlay basis and it comprises the traffic between the broker and all the inner publishers and subscribers. The second is the intra-domain communication which is done on a star topology because Exlap can provide end-to-end communication between the brokers themselves. For a visualisation, see Figure 4.

So, the routing is done inside the domain by one-hub connection between the broker(s) and the clients (subscribers and publishers). The publishing is exclusive to the inter-domain components for security and encapsulation reasons, but subscribing can be internal in the same domain or external using the domain broker which in turn will subscribe to the broker in other domains to get the needed external data.

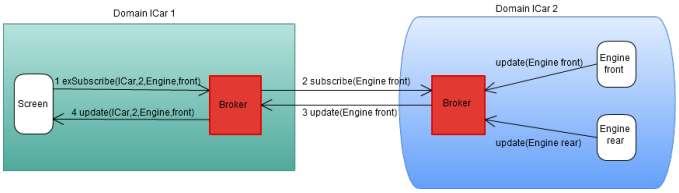

Figure 4. inter- and intra-domain communication

D. Matching

The matching mechanism depends entirely on the types defined in the tree, which describe the contents available in the domain, secure the type safety and provide encapsulation and hierarchical subscription. The type-based and hierarchical pub/sub is not new, as we discussed, but there is no system to our knowledge that addresses the problem of the multi-dimensional representation of the types inside the system. This model we are proposing gives a wide range of options to the designer and implementer.

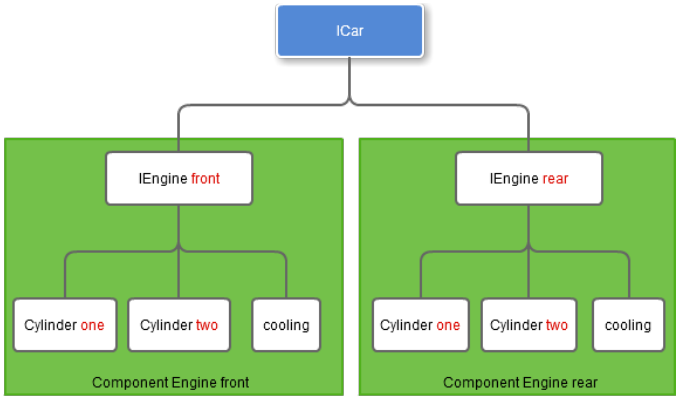

Figure 5. Physical scheme of the car

Specifying the physical dimension is optional and has several degrees, while declaring the type is mandatory. Suppose that we have the previous scheme for the car and the physical scheme in Figure 5, where we have two engines. Each engine has two cylinders and a cooling system. Here, we start with the publishing options; we can assign the cylinder an ID (give it a number) or not, then having two cylinders of the same ID or cylinders without IDs will publish the same content. Now assigning the parent is also optional, in case where no parent is assigned the component will be replicated in all engine components. On the other hand, specifying the parent will give the component unique place in the physical tree. Now subscribing is no less flexible, with the option of using the wild card; e.g., when subscribing to any engine without specifying IDs, it will return all components in the entity and all their children in case of using the wild card. But subscribing to it with ID, will return the corresponding component data only and, in case of using wild card, with its children. Now, if we are interested in just every cylinder, no matter from which engine, then that is also a possibility.

 ISBN: 978-1-61208-610-1

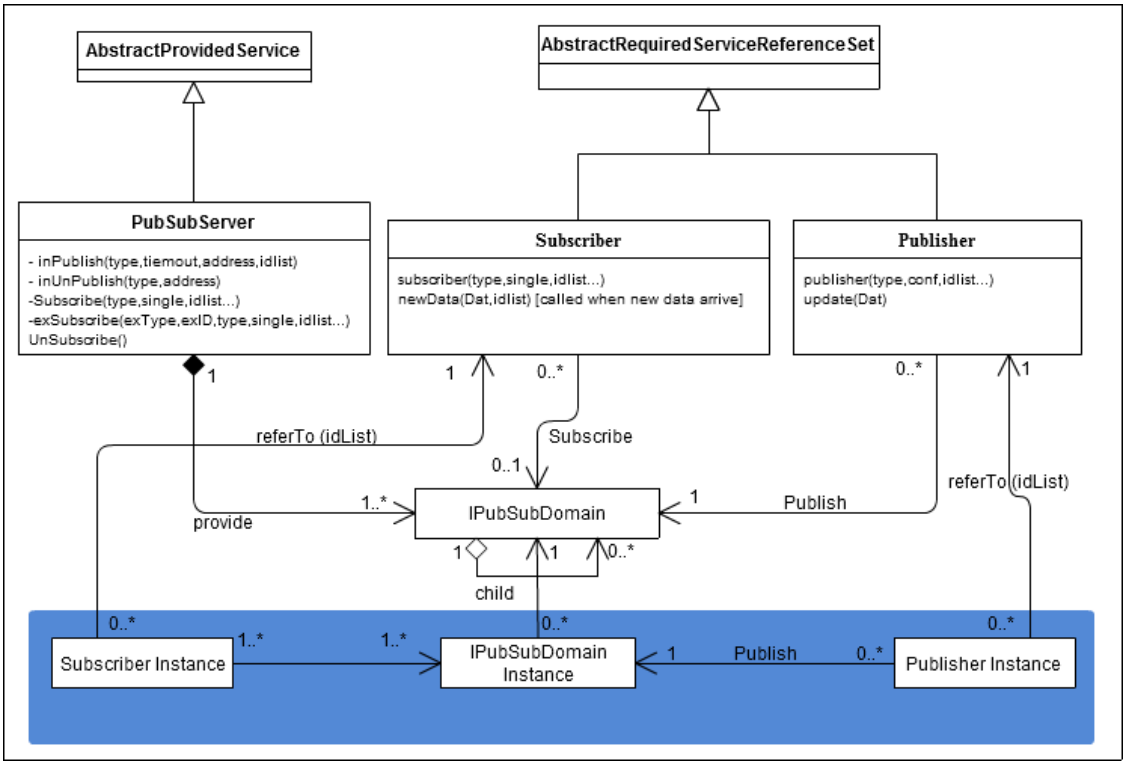

Figure 6. Core elements of the pub/sub system

We notice that once we specify the type there is no wrong answer when it comes to picking the physical dimension, including the use of wild card which implies the functional unity of the embedded components. The architectural details of this system will be discussed in the next section.

IV. System Details

In this section, we present the details of our concept's implementation and the core elements and concepts. For an overview of the core elements and architecture of our system, see Figure 6.

A. Architecture

Since this model is another layer added to DAiSI framework, it extends its framework classes. Review [5] for a complete overview of the used DAiSI frameworks elements. The *AbstractProvidedService* represents the abstract class of any provided service a component needs to offer, so is the *AbstractRequiredServiceReferenceSet* which indicates the services needed by the component. This pub/sub system also offer the broker service on the basis and using the infrastructure of DAiSI. The new element *PubSubServer* works with other normal services and provides the matching and brokering service only for Subscribers and Publishers elements, in other words it works with classic DAiSI. Also the system provides the elements Subscribers and Publishers as extended functionality from *AbstractRequiredServiceReferenceSet* in the same way as the broker.

1) PubSubServer: *PubSubServer* is a core element class of the framework, which provides the main functionality of pub/sub to the whole system. One instance of *PubSubServer* at least is required in every domain. *PubSubServer* depends on the types scheme to function and coin the required types and serve its clients. This service will provide two interfaces; external one for other brokers from other domains, and internal one to handle the internal subscription and publishing requests and real data streaming updates that are pushed to the subscribers.

This class extends the *AbstractProvidedService* and gives the developer the option to feed to this element the types' hierarchical scheme and define parameters like the maximum number of connected clients. The APIs provided by this element is the internal inPublish() and inUnPublish(). the parameters are timeout, which is the time before disconnecting, address is the address of the publisher and the idlist is what defines the physical dimension of the type which we want to publish to. For example, if one wants to publish to the second cylinder of the front engine then one starts with the lowest in the types' tree and makes his way up like that (second front). Subscribing is done automatically when the component in its environment activates a configuration which contains a subscriber. How to initialize the subscribers and publishers will be discussed in the next sections.

2) Publisher: The publisher in DAiSI is a normal *AbstractProvidedService* but with added functionality which are of two folds. The first is to provide a method that can be called to send new data to the broker once it is available. The second is to specify from the types' scheme a type to publish to with specifying the needed idlist. We can specify the idlist or leave it null which can be done on two levels. First, the idlist can be null on the target type itself which means that this hardware is the only one and has no physical semantics

needed for further distinction. The second level is to leave the parents idlist unspecified which means that this component is replicated and embedded at every component that has its parent type no matter what id or physical semantics it holds. Here comes the type-safety to play its role, because it is not necessary to program the embedded systems with vague topic names or types which could lead to untraceable bugs.

3) Subscriber: The Subscriber extends the *AbstractProvidedService* also with added functionality which are: first to provide a method that will be called upon new data arrival, and the second is to specify from the types' scheme a type to subscribe to with specifying the needed idlist, or leave it null for general subscriptions as described before.

4) IPubSubDomain: IPubSubDomain represents the types coined in the implementation. This embody the structure of the types and their corresponding data object which will hold the idlist that represents the physical semantics. Many subscribers can refer to the same component with the same type and physical semantics. Also, publishers can publish to the same component, but this means they will form the same data stream without physical or other kind of distinctions. In this way, we can cover a great variety of use-cases in pub/sub systems.

B. Behavior

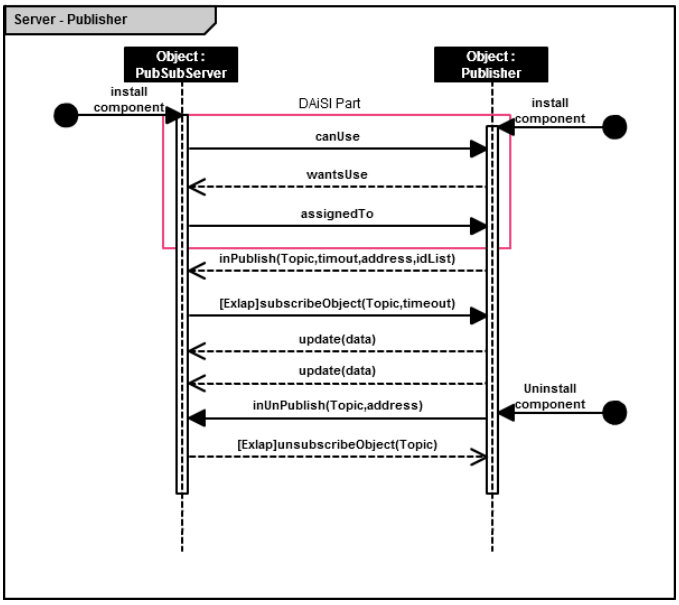

Figure 7. Broker-publisher interaction

The behavior of the system is based exactly on DAiSI components' behavior, where the publisher finds the broker, requests to use its service and when granted the publishing starts. For a visualisation of the data flow, refer to Figure 7. If the component is a subscriber, it will find the broker in the same way, asks for data, and, if granted, it will stream the data using the underlying communication protocol (Exlap in our implementation), see Figure 8. Both sequence diagrams shown are taken from the inter-domain communications.

C. Key Concepts

A key concept in this system is that it offers both pub/sub and the observer pattern communication architecture. Those

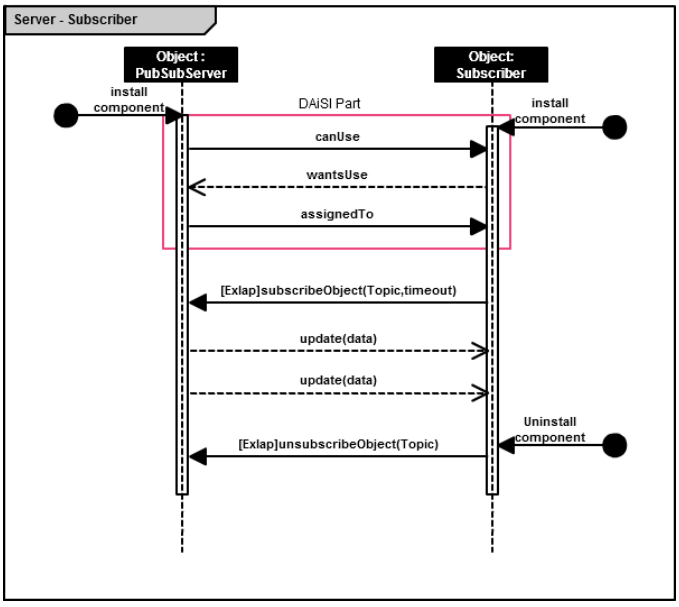

Figure 8. Broker-subscriber interaction

distinctions can vary depending on the domain, but they include things like a one-to-many communication since the observed source has a high level of specificity, so it represents one entity and one only. Like subscribing to the first cylinder of the front engine, this is observing specific information presented by one authorized source. And because of this specificity there is no anonymity between the observer and the observee. On the contrary, we can deduce that when there is anonymity or generality is needed then the streamed data will be formed from several sources or maybe one and the communication will be many-to-many or pub/sub pattern. Figure 9 depicts the type channels and when they can be from single or multiple sources.

Another important concept is the degree of freedom given when using this framework altogether, in which we have the option to leave all the physical dimensions empty and not give the components any physical semantics (idlist). This is the case where we end up with identical types' scheme and real-world components' scheme.

Also, an important feature is that the broker should not need to know in advance the components participating, it only needs the types' scheme, so the components can join, leave or change the physical semantics or type at runtime without affecting the functionality of the whole system.

V. Conclusion

Our work is presented in a specific software architecture that allows for easy replacement of the layers or employing them else-where.

This model is intended for domain-based embedded systems infrastructure that needs to communicate live data asynchronously. We can mention cars, planes, smart houses, surveillance systems and information gathering systems and any autonomous system of which dynamicity depends on its internal data communication including the dynamic adaptive system infrastructure we originally aim to solve the asynchronous communication for.

 ISBN: 978-1-61208-610-1

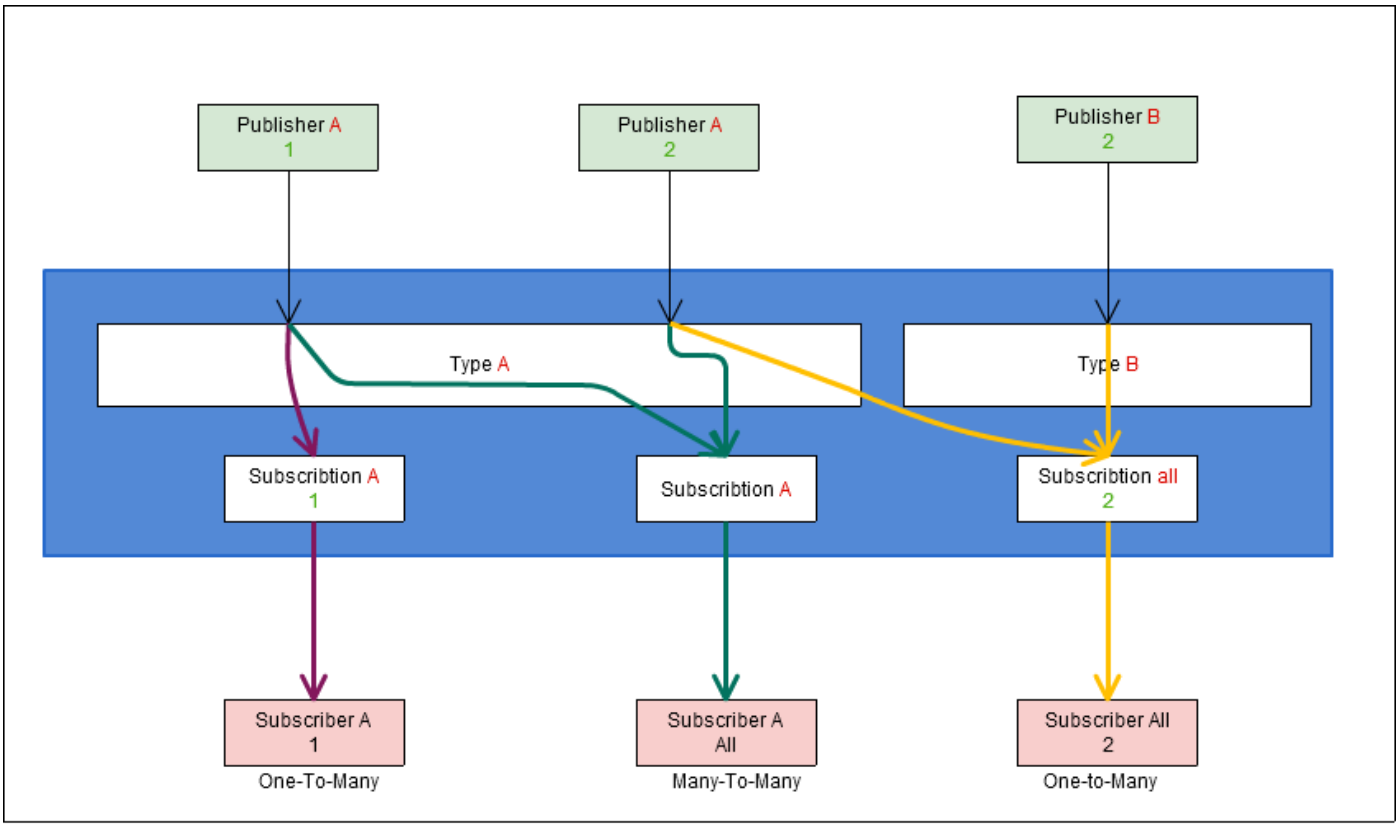

Figure 9. Type Channels

Those domains we talked about like a car for example, can have components of different computational capabilities, so it is critical to move some of the burden away from those components and equip them with an elegant and light-weight solution for their connectivity. Here emerged the idea of type-based pub/sub using a broker, not to mention that the broker provides the decoupling and removes the direct dependencies between the communicating parties in both space time and synchronization [6].

The disadvantage that can be taken on this model is the lack of clear policy that enhances the communication through content filtering mechanisms which can lessen the expressiveness of the model communications and increase the useful data exchanged, which can be a major enhancement in future works. Another great addition would be to extend the physical semantics from including only one feature to maybe group of features the component can be summoned according to.

It is evident in today's standards that the field of Internet of Things (IoT) will have a great share in the researching and industrial community. IoT where machine talks to a machine and exchanges information which helps in taking decisions that sometimes can be critical and need to be quick. Our work focuses on providing the right, scalable, easy to implement and flexible scheme for which these talks can depend on. And that is the key that will open the door for autonomous and smarter systems that exchange categorized data on both the functional and semantical dimensions.

REFERENCES

[1] S. Tarkoma, Publish/Subscribe Systems: Design and Principles, D. Hutchison, S. Fdida, and J. Sventek, Eds. John Wiley & Sons, Ltd, 2012.

[2] L. Zervakis, C. Tryfonopoulos, A. Papadakis-pesaresi, M. Koubarakis, and S. Skiadopoulos, "Full-text Support for Publish / Subscribe Ontology Systems," Proceedings of the 9th Extended Semantic Web Conference (ESWC), Crete, Greece, (postertrack), 2012, pp. 1–2. [Online]. Available: http://arxiv.org/abs/1307.2015

[3] J. Wang, B. Jin, and J. Li, "An Ontology-Based Publish / Subscribe System *," Ifip International Federation For Information Processing, no. 2002, 2004, pp. 232–253.

[4] P. Eugster, "Type-based publish/subscribe," ACM Transactions on Programming Languages and Systems, vol. 29, no. 1, 2007, pp. 6–es. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1180475.1180481

[5] H. Klus and A. Rausch, "DAiSI A Component Model and Decentralized Configuration Mechanism for Dynamic Adaptive Systems," International Journal On Advances in Intelligent Systems, vol. 7, no. 3 and 4, 2014, pp. 27–36.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, no. 2, 2003, pp. 114–131. [Online]. Available: http://portal.acm.org/citation.cfm?doid=857076.857078

[7] Y. Liu and B. Plale, "Survey of publish subscribe event systems," Indiana University Department of Computer Science, no. TR574, 2003, pp. 1–19.

[8] R. S. S. Filho and D. F. Redmiles, "A Survey of Versatility for Publish / Subscribe Infrastructures," Architecture, no. May, 2005, pp. 1–77.

[9] R. Baldoni, L. Querzoni, and A. Virgillito, "Distributed Event Routing in Publish / Subscribe Communication Systems : a Survey," Technical Report, 2005, pp. 1–27.

[10] M. Petrovic, I. Burcea, and H.-A. Jacobsen, "S-ToPSS: Semantic

Toronto Publish/Subscribe System," Proceedings of the 29th international conference on Very large data basesVolume 29, 2003, p. 4. [Online]. Available: http://arxiv.org/abs/cs/0311041

[11] L. I. Pesonen, D. M. Eyers, and J. Bacon, "Access control in decentralised publish/subscribe systems," Journal of Networks, vol. 2, no. 2, 2007, pp. 57–67.

[12] A. Morales, T. Robles, R. Alcarria, and E. Cedeño, "On the support of scientific workflows over Pub/Sub brokers," Sensors, vol. 13, no. 8, 2013, pp. 10 954–10 980.

[13] D. Bickson, E. N. Hoch, N. Naaman, and Y. Tock, "A Hybrid Multicast-Unicast Infrastructure for Efficient Publish-Subscribe in Enterprise Networks," 2009. [Online]. Available: http://arxiv.org/abs/0901.2687

[14] A. Demers, J. Gehrke, M. Hong, and M. Riedewald, "Towards Expressive Publish / Subcribe Systems," Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings, vol. 3896, 2006, pp. 1–18.

[15] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki, "Flexpath: Type-based publish/subscribe system for large-scale science analytics," Proceedings - 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2014, 2014, pp. 246–255.

[16] P. Eugster, R. Guerraoui, and J. Sventek, "Type-Based Publish / Subscribe ."

[17] "EXLAP - Extensible Lightweight Asynchronous Protocol Specification," Tech. Rep. [Online]. Available: https://de.scribd.com/document/158754515/EXLAP-Specification-V1-3-Creative-Commons-BY-SA-3-0-Volkswagen-pdf

 ISBN: 978-1-61208-610-1