A Component Model for Limited Resource Handling in Adaptive Systems

Karina Rehfeldt, Mirco Schindler, Benjamin Fischer and Andreas Rausch

Technische Universitt Clausthal
Clausthal-Zellerfeld, Germany
email: {karina.rehfeldt, mirco.schindler, benjamin.fischer, andreas.rausch}@tu-clausthal.de

Abstract—**Dynamic adaptive systems are systems that change their behavior at run time, based on system, user, environment and context information and needs. System configuration in terms of structure and behavior of open, self-organized systems cannot completely be predicted beforehand: New components may join, others may leave the system, or the behavior of individual components of the system may change over time. These components may compete for limited resources. Especially in Internet of Things (IoT) applications where service consumers directly interact with service providers, the necessity for a fair and lightweight resource access method arises. Therefore, we have elaborated a method which allocates provided services to applications based on a fair and distributed process. Our approach has been implemented on top of our component model called Dynamic Adaptive System Infrastructure (DAiSI).**

***Keywords*–dynamic adaptive systems; decentralized configuration; resource allocation.**

I. Introduction

Software-based systems pervade our daily life at work as well as at home. Public administration or enterprise organizations can scarcely be managed without software-based systems. We come across devices executing software in nearly every household. The continuous increase in size and functionality of software systems has made some of them among the most complex man-made systems ever devised [1].

In the last two decades, the trend towards 'everything, every time, everywhere' has been dramatically increased through a) smaller mobile devices with higher computation and communication capabilities, b) ubiquitous availability of the Internet (almost all devices are connected with the Internet and thereby connected with each other), and c) devices equipped with more and more connected, intelligent and sophisticated sensors and actuators. These trends also pushed research subjects like Internet of Things (IoT) and applications for smart devices, like smart City, smart home or applications in financial and health technology.

Nowadays, these devices are increasingly used within an organically grown, heterogeneous, and dynamic IT environment. Users expect them not only to provide their primary services but also to collaborate autonomously with each other and thus to provide real added additional value. The challenge is therefore to provide software systems that are correct, stable and robust in the presence of increasing challenges such as change and complexity [2]. Especially in the Internet of Things Domain small autonomous devices are expected to interact and collaborate on their own. Nevertheless, the provided services should be stable and reliable.

In open IoT Systems new sensors, actuators and services may enter the system environment at any time and others may leave the system. Hence, it is essential that our systems are able to adapt to maintain the satisfaction of the user expectations and environmental changes in terms of an evolutionary change.

Dynamic change, in contrast to evolutionary change, occurs while the system is operational. Dynamic change requires that the system adapts at run time. Therefore, we must plan for automated management of adaptation. The systems themselves must be capable of determining what system change is required and initiate and manage the change process wherever needed. This is the aim of self-managed systems.

Providing dynamic adaptive systems is a great challenge in software engineering [2]. In order to provide dynamic adaptive systems, the activities of classical development approaches have to be partially or completely moved from development time to run time. For instance, devices and software components can be attached to a dynamic adaptive system at any time. Consequently, devices and software components can be removed from the dynamic adaptive system or they can fail as the result of a defect. Hence, for dynamic adaptive systems, system integration takes place during run time. In our research group, we have for more than ten years developed a framework for dynamic adaptive (and distributed) systems, called Dynamic Adaptive System Infrastructure (DAiSI).

DAiSI is a service-oriented and component based platform to implement dynamic adaptive systems. Components can be integrated into or removed from a dynamic adaptive system at run-time without causing a complete application to fail. To meet this requirement, each component can react to changes in its environment and adapt its behavior accordingly.

At first, it was only possible for components to ask for a special service based on a domain interface they referred to. In [3], we extended the DAiSI component model by the concept of interface roles which takes runtime information in account for the composition and connection of services. With interface roles a domain interface can be enriched. It allows specifying the role of the interface on the basis of runtime information, like the value of a specific parameter.

In DAiSI, the components connect on local optimization views. Each and every component tries to achieve their best local configuration but the resulting overall system configuration might not meet any global optimization goals or fails to meet context requirements. Therefore, we have elaborated an approach to specify context requirements. We introduced the concept of service application specification and component templates in [4]. A service application specification consists of a set of component templates. A template is a placeholder for a set of component with specific properties. The template can be described without knowing individual components. During run time, DAiSI matches existing components to templates autonomously. With this approach an application is build which meets context requirements.

The development of DAiSI was always motivated through running application examples and demonstrators. As DAiSI has been developed for more than ten years, we have demonstrated the application of our approach and our infrastructure in a couple of different research demonstrators and industrial prototypes and products [5] or [6].

Nowadays, IoT-Applications are an emerging field. IoT is different to classic monolithic software systems. Instead of one big application multiple applications for various users are needed. In the EU project BIG IoT [7], an architecture for interoperable IoT-Systems is introduced. The general idea is to use a central marketplace where service and data providers register their service offerings and service consumers are able to search for their required services and data. But to keep the system scalable, the marketplace only takes care of establishing the connection between consumer and provider. If the consumer directly interacts with providers the necessity to control resource access arises. Since there is no central instance to take care of resource management a distributed method is needed. But also, the method to allocate provider resources for consumers should be fair and lightweight.

The goal of this paper is to introduce such a method on top of the DAiSI component model. The rest of this paper is structured as follows: In Section II, we give an overview of relevant related work. Section III gives a short overview of the DAiSI component model with a few hints for further reading. Our extension for limited resource handling is introduced in Section IV, before the paper is wrapped up by a short conclusion in Section V.

II. Related Work

In the field of large-scale systems component-based development is a solid and state-of-the-art approach [8], [9], [10].

In many cases the used framework influences the architectural structure of a system or the other way around a framework is chosen cause of the underlying architecture and its concepts. One example for component based development are middlewares, which not only defines services and establish an infrastructure, but also specifies a component model on top [11]. The CORBA Component Model (CCM) [12] from CORBA [13], a component based middleware, describes different types of communication as synchronous or asynchronous calls by the port type. These ports are characterized in the interface description of the component.

Another example is the middleware DREAM [14], which defines atomic and composed components, so the interconnection between components could be hierarchical. The connection of components takes place at runtime, but it allows only asynchronous communication. The component model of the middleware RUNES [15] allows the dynamic adding and termination of components at runtime, too as CORBA and DREAM. Furthermore it supports the implementation of a separate algorithm, which realize the arrangement of components.

One of the first frameworks, which supports dynamic adaptive reconfiguration was CONIC. A CONIC application was maintained by a centralized configuration manager [16]. Besides it provides a description technique to adapt and modify the structure of the integrated modules of an application. Another framework, building on the knowledge gained through the research in CONIC, was a framework for Reconfigurable and Extensible Parallel and Distributed Systems (REX) [17]. This frameworks defines its own interface description language to specify the interconnection. Components were considered as types, allowing multiple instances of any component to be present at run-time. The framework allowed the dynamic change of the number of running instances and their wiring [18]. Both, the CONIC and REX framework allowed the dynamic adaptation of distributed applications, but only through explicit reconfiguration programs for every possible reconfiguration.

R-OSGi [19] takes advantage of the features developed for centralized module management in the OSGi platform, like dynamic module loading and unloading. It introduces a way to transparently use remote OSGi modules in an application while still preserving good performance. Issues like network disruptions or unresponsive components are mapped to events of unloaded modules and thus can be handled gracefully a strength compared to many other platforms. However, R-OSGi does not provide means to specify application architecture specific requirements. As long as modules are compatible with each other they will be linked. The module developer has to ensure the application architecture at the implementation level. Opposed to that, our approach proposes a high level description of application architectures through application templates that can be specified even after the required components have been developed.

There are many service-oriented approaches and service-orientated Architectures (SOA) [20], which are capable to handle a dynamic behavior. Unknown components can be integrated into it. However, they have the uncomfortable characteristic that the system itself does not care for the dynamic adaptive behavior. The component needs to register and integrate itself. Also, it has to monitor itself whether the used services are still available and adapt its behavior accordingly, if that is no longer the case. But components can be developed independently and reused [21].

In the context of IoT, Stankovic highlighted in [22] eight research topics and challenges. One of them are "Architecture and Dependencies", he mentioned that the sharing of components across simultaneously running applications can result in many systems-of-systems interface problems. The main reason for this is the interaction with actors. A simple example is described in [22] also, imagine a Smart Home system controlling windows, shades and thermostats. If the sensors and actuators are shared between applications, than it could lead to conflicts when these applications have there own assumptions and strategies to modify the room temperature. As shown in [23] this problem occurs always if a resource like an actuator is limited or applications are competing for these resources. This leads to specific infrastructures like DepSys [24] a sensor and actuator infrastructure for smart homes that provides comprehensive strategies to specify, detect, and resolve conflicts.

Anders and Lehner presented a decentralized graph-based approach for agent networks to solve resource allocation problems [25]. Their approach works for structures where you can easily derive such a network like in smart grid systems.

But we are looking at systems where the agents are changing at runtime and where no clear network for resource exchange can be determined.

Therefore we are using a market-based approach like the ones introduced in [26]. Market-based approaches are generally useful because of their simplicity but effectiveness to achieve a sustainable solution by using little information like price and offer and simple interaction like trading. As presented in the next sections our approach builds up on distributed component models and handles the conflicts of limited resources in a general and generic way.

III. DAiSI Component Model

In this section, we want to shortly introduce the existing DAiSi component model. We build our extension on top of the existing component model in section IV. We will use a common example throughout the whole paper which we will introduce next.

Imagine a biathlon training center. The training center consists of a skiing track and a shooting range with several shooting lanes. Biathlon teams are able to train under their trainer's watch. The training center provides a training overview system for each trainer where he can see the current training data of his athletes. For that purpose, each athlete is equipped with at least a pulse sensor. Moreover, a device which measures the currently used skiing technique is attached to the athlete's gear.

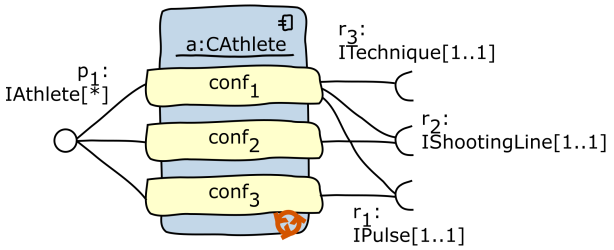

Figure 1. A DAiSI component representing an athlete.

The DAiSI component model is best explained with an example. In Fig. 1, a DAiSI component representing a biathlon athlete is shown. The actual component is the blue rectangle in the background. A DAiSI component consists of different configurations, each of them including one or more provided and required services. The athlete has three different configurations depicted by yellow boxes. Configurations in DAiSI are ordered manually by the designer. The top-most configuration is the best one and therefore the one a component strives to achieve. In Fig. 1, each configuration offers the service IAthlete depicted with a full-circle based on the UML lollipop notation. Accordingly, a required service is depicted by a semicircle. The best configuration in our example requires three different services: ITechnique, a service provided by the skiing technique measuring device; IPulse, a service provided by the pulse device and IShootingLine which is the shooting line evaluating the shooting performance of an athlete.

Fig. 2 shows the DAiSI component model. The different aspects are covered in various papers which were published throughout the years. Therefore, we will stick to a general introduction here and refer to the detailed papers. The orange parts are the extensions introduced in this paper.

The domain architecture of a DAiSI application defines domain interfaces. On the basis of these domain interfaces is decided whether required and provided services can be connected. In [3], the domain interfaces are extended by interface roles. As already mentioned in the introduction interface roles allow the specification of additional constraints for the compatibility of interfaces that use run-time information, bound services and the internal state of a component.

Applications are used to specify context requirements. They narrow down the possible structure of a application configuration. Blueprints for components, so called Templates specify (needed and offered) RequiredTemplateInterfaces and ProvidedTemplateInterfaces which refer to DomainInterfaces and thus form a structure which can be filled with actual services and components by the infrastructure. A more detailed discussion about templates and applications can be found in [4].

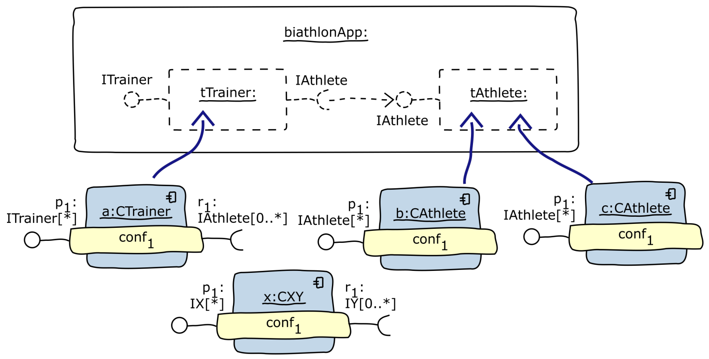

Figure 3. Example for an application with two different component templates.

Fig. 3 shows an example for the usage of templates and applications. biathlonApp specifies an application consisting of two component templates. The first component template tTrainer can be filled with components providing a service referring to the domain interface ITrainer and requiring a service referring to IAthlete. Following this, the second template tAthlete is compatible with components provding an IAthlete service.

With the interface roles and template extension we are now able to describe an application. But in the case of IoT-Domains with many different applications competing for limited resources we have to be able to describe the dependencies between different application instances. In the next section, we will show application scenarios in our biathlon training center introducing our mechanisms and structures for distributed limited resource handling.

IV. Limited Resource Handling on Top of DAiSI

Recall our biathlon example. We have different biathlon teams training in a training center with a limited amount of shooting lanes. Driven by scenarios on top of this example, we will introduce our extensions to the DAiSI component model which were introduced to handle limited resources. The mapping of components to templates and the creation of applications will then no longer be done simply on interface matching criterias but also with regards to resource assignments.

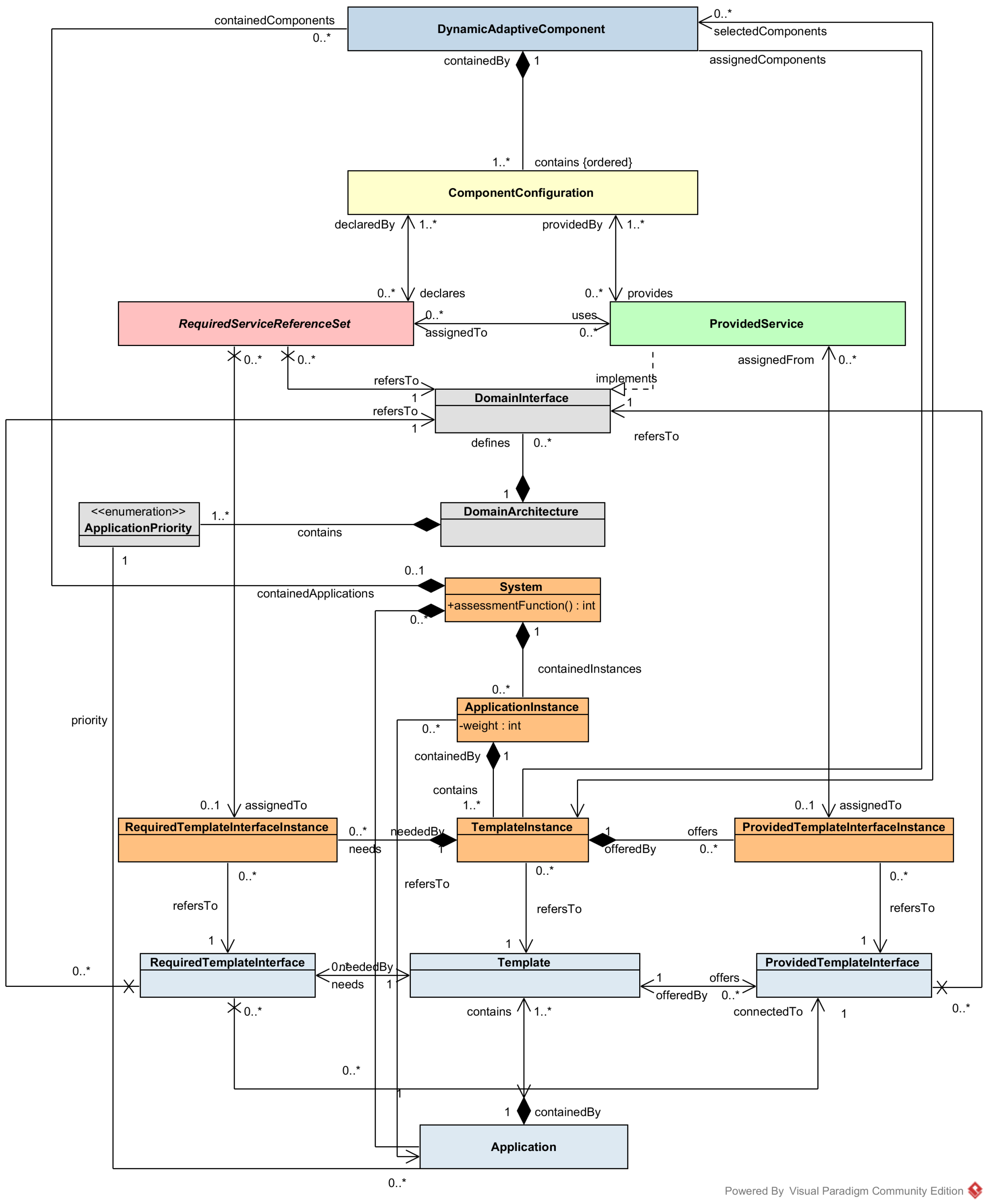

Figure 2. DAiSI component model with Application Instances.

The configuration mechanism of DAiSI which is lengthly introduced in [27], is extended by an agent-based mechanism to broker the association of resources. We will not introduce the technical algorithm here but the component model extension.

A. The Need for Application Instances

Two kind of teams are training in our training center: amateur teams and professional teams. They differ in their configuration and usage of training devices.

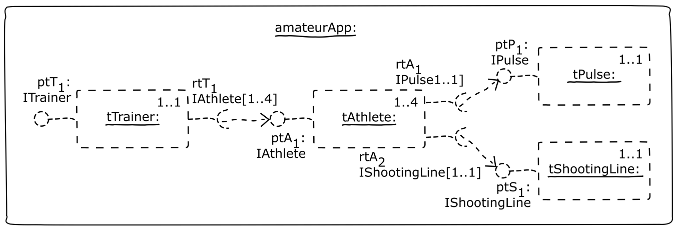

Figure 4. Application for amateur team.

Fig. 4 shows the application and templates for an amateur team. One trainer is training with up to 4 athletes. Each athlete has a pulse measuring device and can use a shooting line. On the other hand, Fig. 5 shows the professional team application. A professional athlete will always use a technique device also.

With the help of our component model until now, we can specify these two application types. But in the training center more than one amateur or professional team might be training. Therefore, the need to introduce an instance level arises. Application and template instances are the first extension made to the DAiSI component model.

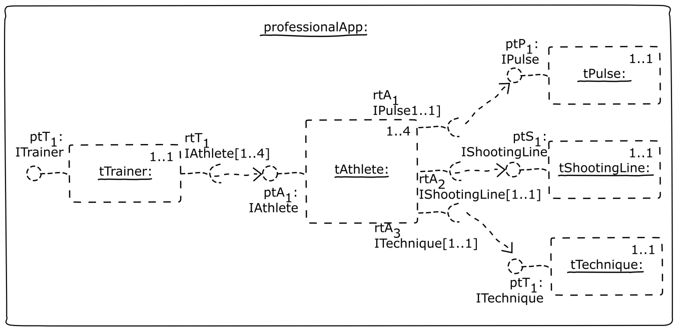

Figure 5. Application for professional team.

The orange parts in Fig. 2 are the extensions made to the component model for limited resource handling. To be able to model a system based on instances, we introduce application instances and template instances. They represent the instance level of our component model. Consequently, the components and provided/required services are no longer bound to the template types but to template instances.

Another new structure is the system. A system is a set of various application instances. It describes the overall configuration of a set of components in these application instances. Each of these application instances may use resources and may even share a resource if the resource allows.

With the help of this extension, we can now describe a system consisting of two amateur team application instances and one professional team application instance.

B. Application Priority

In our example, we have a clearly limited resource: the shooting lanes. Each shooting lane can only be used by one athlete at a time. So the shooting lanes directly influence how many athletes can train on the track. We assume that it is acceptable for a training amateur team to share the shooting lanes. But the professional athletes must have exclusive usage of a shooting lane to train under competitive conditions.

Also, the training of professional athletes is more important, so they should always be preferred to amateur teams. To be able to describe this in our component model, we use ApplicationPriority. ApplicationPriorities are priority classifications for application types. A DomainArchitecture (in our case the biathlon training domain) defines a set of ordered priorities. These priorities are considered by the configuration mechanism when it comes to limited resources. To put it simple, an application type with a higher priority will always be preferred to applications with lesser priority when it comes to limited resources.

Applied to our biathlon example it means that professional team application gets a higher priority than amateur team application. When a professional team wants to use the training center, the assignment of resources will always be in their favor. Application priorities act on type level. But we also need a mechanism for priority on instance level, for example when two different amateur team instances are training. This priority should include run time information because the priority of an application instance may change over time. In the next section, we will introduce our concept and motivate it by another example.

C. Weight

Now that we are able to account for priority on type level, we introduce our concept for priority on instance level. Every ApplicationInstance has a weight. The weight is an indicator for the configuration mechanism how valuable an application instance is for the overall system. During the assignment of resources the weights are used to decide which application ultimately gets the resource.

In our biathlon training center, a training schedule exists. It defines training times for teams. We assign each team instance a weight based on the training schedule. Thereby, we want to make sure that each team may train on their assigned training time but if there are still available shooting lanes in the center, additional teams may train. To be able to achieve that, we assign a team exactly on their training schedule the weight 1. The more the current time differs from their assigned training time, the lower the team's weight gets until it reaches 0. So for instance, until half an hour before their training schedule a team gets the weight 0, a quarter to their training schedule they get the weight 0.5 and exactly on their training schedule they get the weight 1.

Going back to our resource assigning mechanism, if two teams are competing for a shooting lane the team with higher weight, thus closer to their training schedule, will get the assignment of the shooting lane. But a team with weight

0 is also able to get the shooting lane, if no team with a higher weight is asking for it. In the case of same weights, the assignment has to be done randomly. In the end, we have extended our DAiSI component model by an instance level and priorities on type and instance level. With the help of these new features we are now able to handle limited resources on top of DAiSI. It exists a proof-of-concept implementation which will be published in the PhD-Thesis of Benjamin Fischer.

V. CONCLUSION

We introduced an enhancement to our DAiSI component model which allows modeling for limited resource handling. Limited resources are especially a problem in systems with competing applications or shared actuators, for instance IoT systems. To be able to model more than one possible application, which is necessary for IoT systems, an instance level was created. The assignment of resources may be decided on application type level on the basis of application priorities. Additionally, weights are used on application instance level to model the significance of an application instance to the overall systems.

Klus et. al [4] presented a configuration algorithm to assign components to applications. The introduced enhancement of the component model in this paper may be used in an extended configuration algorithm which also deals with the assignment of limited resources. A possible implementation is conceived and will be published in the PhD-Thesis of Benjamin Fischer.

REFERENCES

[1] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan et al., “Ultra-large-scale systems: The software challenge of the future,” DTIC Document, Tech. Rep., 2006.

[2] J. Kramer and J. Magee, “A rigorous architectural approach to adaptive software engineering,” Journal of Computer Science and Technology, vol. 24, no. 2, 2009, pp. 183–188.

[3] H. Klus, D. Herrling, and A. Rausch, “Interface Roles for Dynamic Adaptive Systems,” Proceedings of ADAPTIVE, 2015, pp. 80–84.

[4] H. Klus, A. Rausch, and D. Herrling, “Component Templates and Service Applications Specifications to Control Dynamic Adaptive System Configurations,” in AMBIENT 2015, The Fifth International Conference on Ambient Computing, Applications, Services and Technologies, vol. 5. Nice, France: IARIA, Jul. 2015, pp. 42 – 51.

[5] A. Rausch, D. Niebuhr, M. Schindler, and D. Herrling, “Emergency management system,” in Proceedings of the International Conference on Pervasive Services 2009 (ICSP 2009), 2009.

[6] C. Deiters, M. Köster, S. Lange, S. Lützel, B. Mokbel, C. Mumme, and D. Niebuhr, “Demsy-a scenario for an integrated demonstrator in a smartcity,” NTH Computer Science Report, vol. 1, 2010.

[7] B. I. project. Bigiot - bridging the interoperability gap of the internet of things. [Online]. Available: http://big-iot.eu/ (2016)

[8] C. Szyperski, Component Software: Beyond Object-Oriented Programming (2nd Edition), 2nd ed. Addison-Wesley Professional, 2002. [Online]. Available: http://amazon.com/o/ASIN/0201745720/

[9] A. MacCormack, J. Rusnak, and C. Y. Baldwin, “The impact of component modularity on design evolution: Evidence from the software industry,” SSRN Electronic Journal, 2007.

[10] B. Councill and G. T. Heineman, “Definition of a software component and its elements,” Component-based software engineering: putting the pieces together, 2001, pp. 5–19.

[11] P. A. Bernstein, “Middleware: a model for distributed system services,” Communications of the ACM, vol. 39, no. 2, 1996, pp. 86–98.

[12] N. Wang, D. C. Schmidt, and C. O’Ryan, “Overview of the corba component model: Component-based software engineering,” G. T. Heineman and W. T. Councill, Eds. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 2001, pp. 557–571. [Online]. Available: http://dl.acm.org/citation.cfm?id=379381.379581

[13] Object Management Group - OMG, “Corba component model specification,” 2006. [Online]. Available: http://www.omg.org/spec/CCM/4.0/PDF

[14] M. Leclercq, V. Quéma, and J. Stefani, “Dream: a component framework for constructing resource-aware, configurable middleware,” IEEE Distributed Systems Online, vol. 6, no. 9, 2005, p. 1.

[15] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, and S. Zachariadis, “The runes middleware: A reconfigurable component-based approach to networked embedded systems,” in 2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications, vol. 2, 2005, pp. 806–810.

[16] J. Magee, J. Kramer, and M. Sloman, “Constructing distributed systems in conic,” IEEE Transactions on Software Engineering, vol. 15, no. 6, 1989, pp. 663–675.

[17] J. Kramer, J. Magee, M. Sloman, and N. Dulay, “Configuring object-based distributed programs in rex,” Software Engineering Journal, vol. 7, no. 2, 1992, pp. 139–149.

[18] J. Kramer, “Configuration programming-a framework for the development of distributable systems,” in COMPEURO’90: Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering-Systems Engineering Aspects of Complex Computerized Systems. IEEE, 1990, pp. 374–384.

[19] J. S. Rellermeyer, G. Alonso, and T. Roscoe, “R-osgi: distributed applications through software modularization,” in Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware. Springer-Verlag New York, Inc., 2007, pp. 1–20.

[20] H. Li and Z. Wu, “Research on distributed architecture based on soa,” in 2009 International Conference on Communication Software and Networks, pp. 670–674.

[21] M. Turner, D. Budgen, and P. Brereton, “Turning software into a service,” Computer, vol. 36, no. 10, 2003, pp. 38–44.

[22] J. A. Stankovic, “Research directions for the internet of things,” IEEE Internet of Things Journal, vol. 1, no. 1, 2014, pp. 3–9.

[23] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, “Supporting concurrent applications in wireless sensor networks,” in Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, ser. SenSys ’06. New York, NY, USA: ACM, 2006, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182822

[24] S. Munir and J. A. Stankovic, “Depsys: Dependency aware integration of cyber-physical systems for smart homes,” in 2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), pp. 127–138.

[25] G. Anders and P. Lehner, “Self-Organized Graph-Based Resource Allocation,” in Self-Adaptive and Self-Organizing Systems (SASO), 2016 IEEE 10th International Conference on, Sept 2016.

[26] S. H. Clearwater, Market-based control: A paradigm for distributed resource allocation. World Scientific, 1996.

[27] H. Klus, A. Rausch, and D. Herrling, “DAiSIDynamic Adaptive System Infrastructure: Component Model and Decentralized Configuration Mechanism,” International Journal On Advances in Intelligent Systems, vol. 7, no. 3 and 4, 2014, pp. 595 – 608. [Online]. Available: http:sse-world.deindex.phpdownloadfileviewinline370