HAL
open science

Adaptive Exchange of Distributed Partial Models@run.time for Highly Dynamic Systems

Sebastian Gotz, Ilias Gerostathopoulos, Filip Krikava, Adnan Shahzada, Romina Spalazzese

To cite this version:

Sebastian Gotz, Ilias Gerostathopoulos, Filip Krikava, Adnan Shahzada, Romina Spalazzese. Adaptive Exchange of Distributed Partial Models@run.time for Highly Dynamic Systems. Proceedings of 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, May 2015, Firenze, Italy. hal-01119490

HAL Id: hal-01119490

https://inria.hal.science/hal-01119490v1

Submitted on 12 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Adaptive Exchange of Distributed Partial Models@run.time for Highly Dynamic Systems

Sebastian Götz*, Ilias Gerostathopoulos†, Filip Krikava‡, Adnan Shahzada§ and Romina Spalazzese¶
*Technische Universität Dresden, Germany, Email: sebastian.goetz@acm.org
†Charles University in Prague, Czech Republic, Email: iliasg@d3s.mff.cuni.cz
‡INRIA Lille / University of Lille 1, France, Email: filip.krikava@inria.fr
¶Malmö University, Sweden, Email: romina.spalazzese@mah.se
§Politecnico di Milano, Italy, Email: adnan.shahzada@polimi.it

***Abstract*—Future software systems are characterized by their high dynamicity. For example, we are experiencing a world where Cyber-Physical Systems (CPSs) play a more and more crucial role. CPSs integrate computational, physical, and networking elements; they comprise a number of subsystems, or entities, that are connected and work together. The open and highly distributed nature of the resulting system gives rise to unanticipated runtime management issues such as the organization of subsystems and resource optimization.**

In this paper, we focus on the problem of knowledge sharing among cooperating entities of a highly distributed and self-adaptive CPS. Specifically, the research question we address is how to minimize the knowledge that needs to be shared among the entities of a CPS. If all entities share all their knowledge with each other, the performance, energy and memory consumption as well as privacy are unnecessarily negatively impacted. To reduce the amount of knowledge to share *between CPS entities*, we envision a *role-based adaptive knowledge exchange technique* working on *partial runtime models*, *i.e.*, models reflecting only part of the state of the CPS. Our approach supports two adaptation dimensions: the *runtime type* of and *conditions* over the knowledge. We illustrate the feasibility of our technique by discussing its realization based on two state-of-the-art approaches.

I. Introduction

Currently, we are experiencing an increasingly open world where more and more distributed, highly dynamic systems, as for example Cyber-Physical Systems (CPSs) [1], [2], [3], play a key role. CPSs integrate computational, physical, and networking elements that might perform many different tasks within disparate contexts as for instance smart living spaces. CPSs are composed of *multiple subsystems* or *entities* that have different goals and different *partial views* of the world. Subsystems can communicate and form or participate in *temporary ad-hoc collaborations* with one another and any subsystem can freely arrive and leave the environment at any time. The open and highly distributed nature of these systems gives rise to unanticipated runtime management issues such as the organization of subsystems and resource optimization.

A promising approach to manage these issues is to develop such systems as self-adaptive systems [4] by using self-organization and self-optimization techniques. Each subsystem, then, becomes autonomous and makes decisions based on its own knowledge while taking into account the invariants of the whole system and its environment [5]. Consequently, each subsystem has its own knowledge exchange policy and there is no central point of coordination from which the complete knowledge of the system would be accessible.

This work focuses on a challenging problem for distributed self-adaptive systems: *how to minimize the knowledge that needs to be shared and synchronized among the entities of such a system*. We do not address the problem of integrating systems not developed to interoperate with each other. Since the subsystems hold only a partial view of the complete system, local decision making requires them to exchange their knowledge to synchronize their partial views, which might be a costly operation. A simplistic and naive solution would be that all entities share all their knowledge with each other. This has several drawbacks, which our approach aims to overcome:

- *Performance.* The intense, unnecessary knowledge exchange decreases system performance, as it leads to unnecessary computing cycles and network saturation.
- *Energy Consumption.* The more knowledge is to be exchanged, the more energy is required for it, but the capacity of the participating subsystems is typically restricted.
- *Memory Consumption.* CPSs typically comprise small devices, which are limited in terms of the memory. Hence, storing superfluous knowledge is not a viable option.
- *Privacy.* The subsystems do not necessarily belong to the same owner or come from the same vendor. In such cases, they might possess knowledge that cannot be shared without threatening the privacy of the owner.

In this paper, we present a *role-based adaptive knowledge exchange technique* working on *partial runtime models* of highly dynamic systems. A partial runtime model reflects the partial view of an entity of such a system representing only a part of the system's state. Our technique supports adaptation with respect to two dimensions: the *runtime type* of and *conditions* over the knowledge. Concretely, the contributions of this paper are:

(i) *three strategies* for knowledge exchange between collaborating subsystems with tradeoffs (*cf.* Section IV-A),
(ii) *role-based runtime models* to specify the partial views of the subsystems (*cf.* Section IV-B),
(iii) *an illustration* of the role-based adaptive knowledge exchange for the partial runtime views in ad-hoc collaborations (*cf.* Section IV-C).

The role-based knowledge exchange allows the collaborating subsystems to reduce the amount of knowledge they exchange. The overall technique faces a tradeoff between the likelihood of optimal decision making w.r.t. the individual goals and the associated cost, which both vary in terms of the amount of knowledge exchanged. By facilitating the exchange of the minimum pertinent knowledge, our technique helps to improve the overall CPS performance, energy and memory consumption, as well as reducing privacy threats whilst assuring good local decision making. We discuss two possible realizations of our approach through two different state-of-the-art technologies.

The remainder of the paper is organized as follows. Section II describes a motivating, running example. Section III outlines the concepts of models@run.time and role-based modeling, which our technique is based on. Our approach to knowledge exchange in distributed partial runtime models is presented in Section IV. Section V discusses related work. To show the feasibility of our approach, two realizations of our approach are outlined in Section VI. Finally, Section VII draws conclusions.

II. The Cleaning Robots Running Example

A cleaning company uses *cleaning robots* to carry out cleaning jobs on a variety of premises. After office hours, the company deploys groups of cleaning robots on contracted sites with the objective to efficiently clean the space before a certain deadline. A site can have multiple floors with different layouts. As the cleaning robots are often changing cleaning sites, it would be impractical to assume that these sites have smart space capabilities or infrastructure. Lacking the static infrastructure that would play the central point that could provide the global view of the environment and coordinate the cleaning job, the robots need to be autonomous and independent. They are equipped only with a short-range communication device (*e.g.* Bluetooth) allowing them to exchange data while being in proximity of one another.

There are various types of robots (*e.g.* specialized in dry or wet cleaning) with different specifications. During its operation, each robot maintains information about its own state (*e.g.*, battery level, bin capacity) and about its environment (*e.g.*, floor plan, the parts of the floor it has cleaned) in the form of runtime models. While the robots only acquire a partial knowledge about their environment, they can communicate together and query each others' runtime models in order to extend their knowledge and therefore improve their decisions. For example, a robot might ask for the parts of the floor that have already been cleaned by others in order to avoid revisiting the same places. However, it shall only require such information in the case it is in the "cleaning" state and not *e.g.* going for charging. It should also only request it from the right robot type. Furthermore, it should limit the query based on its available resources and skip places unreachable within its current battery level. Finally, knowledge exchange should be reduced as much as possible to lower the energy impact.

Essentially, the described scenario is about the exchange of runtime models that are both distributed and partial. The above example helps to explain that the amount of knowledge exchanged should be adaptable at runtime depending on: (i) which entities communicate, *i.e.*, based on the type of the runtime model; (ii) the state of the requester.

An adaptive knowledge exchange technique for runtime models can therefore be regarded as a means that allows developers to express knowledge exchange between runtime models and automate its adaptation at runtime, based on the runtime type of knowledge and conditions over it.

III. Background

In this section we give a brief summary of models@run.time and role-oriented modeling, which our approach is based on.

Models@run.time. The change from programming stationary to mobile devices (*e.g.*, CPSs) introduced several new challenges on software engineering techniques. Among them is the necessity to handle contextual changes in applications. For example, the change in brightness when a car drives through a tunnel should lead to a color inversion of the navigation display. In other words, software for mobile devices is required to reflect on itself and its environment.

To enable this self- and context-awareness, the systems have to keep the respective information. The models@run.time paradigm proposes to use modeling techniques to capture and process this information using runtime models, *i.e.*, models representing a view on the current state of the system [6], [7].

Our approach leverages models@run.time since each autonomous subsystem keeps and updates its own runtime model. Our approach extends the existing work in the models@run.time community by discussing three adaptive knowledge exchange strategies to synchronize the partial runtime models of subsystems forming ad-hoc collaborations.

Role-based Modeling. The modeling of complex, dynamic domains has been a key challenge of software engineering for more than 40 years. Nowadays, a huge amount of software is developed according to the object-oriented paradigm, which is based on concepts introduced already in 1967 [8].

Role-oriented modeling, originally introduced by Bachman in 1973 [9], is an alternative, which provides means to capture the contextual and relational nature of objects, *i.e.*, allows one to model self- and context-aware software [10].

Roles differ from objects in terms of the properties of their types [11]. An object, being an instance of a class, is rigid and non-founded, meaning that the object cannot change its type and is not required to depend on any other object. A role, being an instance of a role type, is anti-rigid and founded, *i.e.*, can change its type at runtime, but has to be connected to an object, which is playing this role. For example, *Student* is a role type, because a person can stop to be a student without ceasing to exist. It is the person which plays the role of a student. The type *Person*, on the contrary, is a class. Moreover, the focus in role-based modeling is on collaborations. Models covering classes and role types are called role models and can be seen as objectified collaborations.

Our approach leverages on role-based modeling by superimposing role models on runtime models of autonomous entities, which allows to dynamically type the runtime information.

IV. Knowledge Exchange in Distributed Partial Runtime Models

In this section, we present our technique for role-based adaptive knowledge exchange between collaborating CPS. First, we introduce three strategies for knowledge exchange characterized by different tradeoffs (*cf.* Section IV-A). Then, by exploiting our cleaning robots example, we present our role-based runtime models (*cf.* Section IV-B). Finally, we illustrate the role-based adaptive knowledge exchange of partial runtime models through a query language (*cf.* Section IV-C).

A. Knowledge Exchange Strategies

Towards the definition of knowledge exchange strategies, there are two aspects that are relevant: the *degree of pairing among subsystems* for their communication and the *amount of knowledge* to be shared among them. The first can be either *total*, *i.e.*, every subsystem communicates with each other, or *partial*, *i.e.*, only some subsystems communicate with some others. The amount of knowledge, on the other hand, can be either *complete*, *i.e.*, a subsystem exchanges all its knowledge, or a *subset*, *i.e.*, a subsystem exchanges only a subset of its knowledge, Consequently, we identified the following three strategies that each subsystem can use to exchange its knowledge with the other subsystems.

1) *Total-Complete*. Each subsystem exchanges its complete knowledge with every subsystem. In this case each subsystem maintains a complete view of the whole system.
2) *Partial-Complete*. Each subsystem exchanges its complete knowledge, but only with the subsystems it directly collaborates with.
3) *Partial-Subset*. Each subsystem exchanges only part of its knowledge, and only with the subsystems it collaborates with. The knowledge part that is being exchanged pertains to the current collaboration.

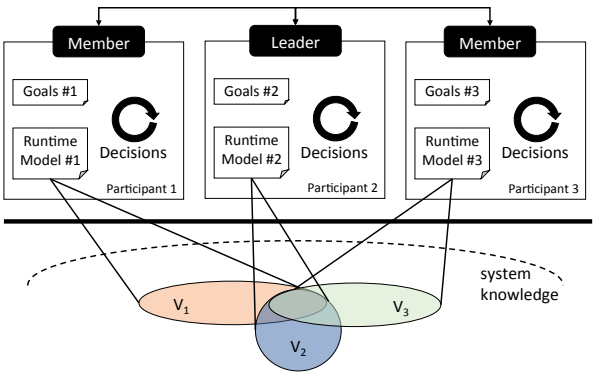

Fig. 1. An Ad-hoc Collaboration Among Three Subsystems.

Figure 1 depicts three subsystems (*Participant 1, 2 and 3*) forming an ad-hoc collaboration. The collaboration is represented by the black filled boxes specifying the role (*Member* or *Leader*) of each participant in the collaboration. Leaders directly collaborate with members, but members do not collaborate with each other directly (only through the leader). Each subsystem has: its own knowledge, captured in a *Runtime Model* (RM), its own *Goals*, and its own *Decision* making mechanisms. A RM specifies a set of relevant aspects of the world and they capture only a part of the whole *System Knowledge* (*i.e.*, V_1, V_2, V_3 that are partial views). Since each subsystem has its own RM, this shall be updated as frequently as needed to adequately reflect the current world.

When some subsystems form a collaboration, their RMs will overlap as they will capture some common parts of the system knowledge. Since each subsystem potentially employs different abstraction mechanisms (masking) on the observed knowledge, it is highly improbable for two subsystems to have identical partial RMs. Each RM is the result of applying an abstraction function f_i on the part of the world V_i covered by the system: $RM_i = f_i(V_i)$.

Below, we revisit and elaborate on the three identified strategies for knowledge exchange.

First strategy (Total-Complete). In the first strategy, where all subsystems share all their knowledge, all subsystems will base their decisions on $\bigcup_{i \in 1..n} RM_i$, where n is the number of participants. This, in turn, offers the *benefit* that all participants are guaranteed to have all required knowledge for their local decision making. The *downside* is the negative impact on performance, memory and energy as well as privacy, because potentially more knowledge than necessary is exchanged.

Second strategy (Partial-Complete). For the second strategy only directly collaborating subsystems exchange their complete knowledge. Knowing the application-specific collaboration, allows to reduce the knowledge exchange. Participant 1 shares its knowledge with Participant 2, but not with Participant 3. Similarly, Participant 3 only shares its knowledge with Participant 2. Only Participant 2 has the knowledge of all participants as it directly collaborates with all others. Thus, the *drawbacks* of the first strategy are *reduced*. On the contrary, the *local decisions* of the participants can be *suboptimal*, due to not having the complete knowledge.

Third strategy (Partial-Subset). Finally, in the third strategy, each subsystem exchanges only part of its knowledge and only with the subsystems it directly collaborates with. As subsystems can take part in multiple collaborations, the challenge is to identify which knowledge of which participant is actually required for their decision making. This strategy allows the subsystems to cherry-pick which information is exchanged in order to further *decrease* the amount of knowledge that is transferred. While it might *decrease* the *optimality* of the local decision making, this strategy helps to *improve* performance, memory and energy consumption as well as privacy.

The third strategy can therefore vary the tradeoffs between optimality of local decision making, which can be seen as the utility of the participants, and the respective cost. For this, the optimal tradeoff in terms of utility and cost for local decision making is to be identified, where both utility and cost, are functions over the amount of knowledge to be exchanged. To reach this goal, we propose to leverage role models, which

are objectified collaborations. The binding of roles to players enables the identification of knowledge required for decision making based on the runtime type of that knowledge.

B. Role-based Runtime Models

Figure 2 depicts in the upper-left the *role model* Cleaning that is comprised of five role types: a cleaner, a spot, a clean and dirty spot and a full cleaner role type. The links between the role types are role constraints as introduced by Riehle [12].

For example, the implication constraint between the clean spot and spot denotes the necessity of an object playing the clean spot role to play the spot role, too (this holds similarly for dirty spot and spot). The prohibition constraint between the clean spot and dirty spot denotes the necessity for an object playing one of the two roles, not to play the other role at the same time. The solid arrow between cleaner and spot specifies the ability of the cleaner to work on the spot (*i.e.*, is a role-use/association constraint). Note that each role type can define its own attributes, as shown for the cleaner role type. Consequently, role instances can have a state. The binding between roles and their players is expressed using the *fills* constraint between role types and classes. In the example, fields can play spot roles and robots can play cleaner roles. Moreover, the Cleaning role model comprises a nested role model for rooms. Thus, rooms are modeled as a collaboration of spots, which enables the movement of walls in office buildings. The annotation {1+} is a multiplicity constraint on the number of role instances required by an instance of the role model to be valid (in the example capturing the fact that a room consists of at least one spot).

A concrete scenario is depicted on the right side of Figure 2. Here the world consists of three fields, where two play the clean spot role and one the dirty spot role. In addition, one robot exists, which is participating in the collaboration in the role of a cleaner. The specification of when a player starts or stops to play a role is part of the description of the role types. For example, the cleaner role type will initiate the playership of the full cleaner role for its associated player, when it senses that it is full, *i.e.*, whilst performing its cleaning behavior.

To illustrate how to distinguish collaboration-specific knowledge, which is to be identified for the third knowledge exchange strategy, Figure 2 additionally depicts the Disposal role model. This additional role model specifies that robots can play the role of being bin lorries with a filling level and fields can play the role of bins, with their own filling level. The bin role type denotes an empty field where the cleaning robots are supposed to collect all the trash. If a robot is currently seeking for a bin, it only requires the knowledge captured by the Disposal, but not by the Cleaning role model.

In general, the specification of collaboration-specific knowledge is realized by modeling multiple collaborations, where role types can be played by the same entities of the runtime model. If two autonomous systems approach each other and intend to exchange knowledge, they are now enabled to query collaboration-specific knowledge by formulating the queries against the role models instead of the runtime model.

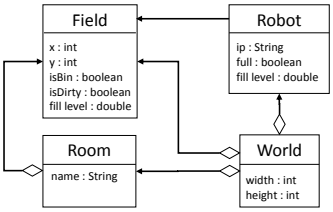

Fig. 3. Metamodel of Runtime Model Without Role Models.

For a comparison between our proposed approach of using role models superimposed on runtime models, Figure 3 depicts a possible metamodel for the runtime model of the above example without role models, *i.e.*, comprising all knowledge– which was extracted to the two role models in Figure 2.

C. Illustration

Imagine the scenario where two cleaning robots approach each other and one is seeking for the next dirty spot to clean. In the case without role models, the query could look as follows:

```
SELECT f.* FROM Field f WHERE f.isDirty = true
```

As a result, the robot will get all the field objects that the other robot knows of being dirty. But, these objects contain additional, superfluous knowledge, which in this case is even known in advance: the field cannot be a bin and, hence, the fill level is set to a default value. In contrast, the query against the role model of Figure 2 could look, in RSQL [13], as follows:

```
SELECT * FROM Cleaning PLAYING DirtySpot
```

The **PLAYING** clause refines the selection based on role types. This query will return all dirty spot role instances that the other robots know of in its current cleaning collaboration. These instances provide just the required knowledge to the requesting robot: the property of the field to be dirty and its coordinates. The knowledge about disposal is not shared.

Thus, using role models, collaborating entities can share knowledge based on the runtime type of the data, *i.e.*, a type that can change at runtime.

An important benefit of using role models to reduce the amount of knowledge to be exchanged is increased evolvability. The restriction to receive only the coordinates of dirty spots could be realized by explicitly naming them in the projection of the SQL query. That is for the example above:

```
SELECT f.x, f.y FROM Field f
  WHERE f.isDirty = true
```

The problem of explicitly naming the attributes of interest arises when the system evolves and new attributes are added to the field. For example, adding a third coordinate (z) to the fields. Without role models, all queries restricting the knowledge to the coordinates of the field have to be adjusted, whereas with role models all queries can stay unchanged.

Next to this, it is of course also possible to filter the results by conditions using the SQL language construct **WHERE**. For example, a robot seeking for the next dirty spots could narrow

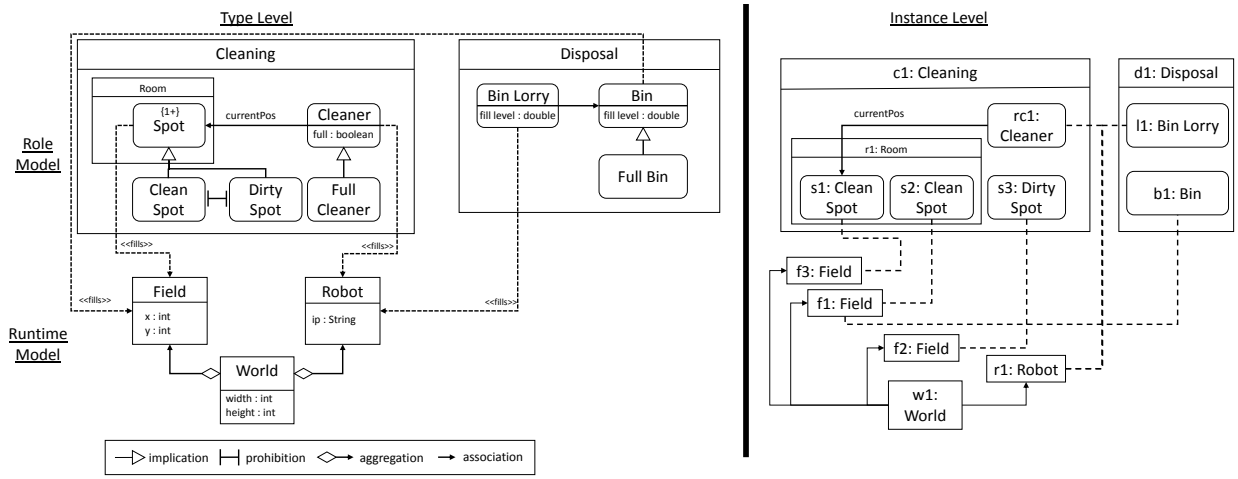

Fig. 2. Combination of Role Models and Runtime Models.

the search to include only the tiles reachable before it has to recharge (remDist is the remaining travel distance):

```
SELECT * FROM Cleaning PLAYING DirtySpot ds
  WHERE sqrt((x-ds.x)^2 + (y-ds.y)^2) < remDist
```

Consequently, role models allow to reduce the amount of knowledge to be shared by dynamically changing the runtime type of the requested objects and by evaluating conditions.

V. Related Work

The problem of execution and knowledge exchange via partial models across different elements in the system has been studied by many researchers from multi-agent systems, component-based systems, and nature-inspired computing disciplines. The proposed approaches differ in terms of (i) how they *organize* the *system* and *coordinate* the *partial models* to achieve system-level goals, and (ii) the *mechanisms* they employ for *knowledge exchange* and *collaboration*. Hence, we limit our discussion to these criteria.

From an organizational point of view, we find grouping or clustering of components/agents to be a popular approach. For instance, DEECo [14], SeSaMe [15] and ASPECS [16], [17], form interactions and collaborations between different entities of a system through ensembles, groups and holarchies, respectively. DEECo is an ensemble-based component system where an ensemble represents dynamic binding of a set of components and thus determines their composition and interaction. Each component has a state and corresponding processes that are independent of the ensembles it is part of. SeSaMe coordinates distributed components in various self-organizing inter-composed groups based on the types of roles they can play. ASPECS is an agent-oriented methodology that employs holonic organization of agents to map system capabilities to user injected goals. A holon is a general term denoting an entity that has its own individuality, but, at the same time, is embedded in larger wholes.

SAPERE [18], on the other hand, takes a nature-inspired approach for component organization and binding, based on the concept of chemical tuple spaces. Each component in the SAPERE ecosystem has an associated semantic representation (chemical annotation); this enables dynamic component interactions in an unsupervised way. The components interact with each other according to the so called "eco-laws" to fulfill their own goals as well as the goals of the overall ecology.

For communication, the proposed solutions adopt different paradigms that include direct messaging, ad-hoc collaboration spaces, and the use of mediator models [19], [20]. Typically, agent-based systems use direct messaging among the participating partial models to enable negotiation and persuasion between agents. ASPECS conforms with the FIPA protocol [21]. SeSaMe also incorporates direct messaging among the components. SAPERE, on the other hand, makes use of a distributed shared space, where context and knowledge are provided by a set of semantic representations of agents. Another approach [22] generates a mediator model to bind systems represented in a labeled transition system.

The amount of knowledge, exchanged at runtime between components, each having a partial view of the system, is significant and influences the efficiency of the underlying system. In DEECo, the knowledge of every component is proactively shared among all components and only used in components of the same ensemble (the ensemble membership condition is evaluated locally in each component) [23]. SeSaMe handles knowledge distribution through dynamic roles that restrict the amount of knowledge to be shared among interacting components. SAPERE and ASPECS handle this by matching service providers (capabilities) and consumers (goals).

VI. Discussion

In this section, we provide a discussion about possible realizations of our approach. We focus on two promising approaches surveyed in the related work, namely DEECo and SeSaMe, to analyze how they can employ different proposed strategies and use the proposed role models to guide the design process while developing highly dynamic systems.

a) Realization of First Strategy through DEECo: To deal with extreme dynamicity coming from external uncertainty in CPS environments (intermittent connections, physical mobility of nodes, opportunistic communication schemes, etc.), DEECo employs the first strategy (Total-Complete) from our list of knowledge exchange strategies. DEECo provides the component-based abstractions and corresponding machinery for implementation and deployment of CPS applications which fit our view of autonomous collaborating subsystems. In particular, the illustrative example from Section IV-C corresponds to a DEECo application comprising two DEECo components: Robot and Room. The Room contains a map of several spots. It also comprises a dynamic collaboration group (*ensemble* in DEECo terms): DirtySpotsExchange. Its goal is to exchange the knowledge of the Room, specifically the set of the remaining dirty spots in the room, with the Robot.

In particular, every node in a DEECo application periodically and proactively gossips both its own and the knowledge of other nodes that it knows of with all other nodes[1] [23]. In our example, this means that all the knowledge of the Room is passed to the Robot, and vice versa. Then, ensemble evaluation is performed *locally* on the Robot, which results into mapping data from its temporary knowledge base (used by the gossip protocol) to its internal knowledge, a step that finally makes the dirty spots (pertinent to this collaboration) accessible from within the Robot's processes.

Thus, the knowledge exchange approach favored by DEECo trades having complete knowledge for local decision-making for higher network utilization and data redundancy.

b) Realization of Second Strategy through SeSaMe: SeSaMe employs the second strategy (Partial-Complete) as it uses group-based coordination to form collaborations among heterogeneous subsystems. The groups are formed on the basis of roles that various components are capable to perform. Therefore, the proposed role model can be used to guide the design process in SeSaMe. A SeSaMe application of the running example comprises two component types: Robot and Field. Each Robot maintains its own state including its location. The Robot component can play the Cleaner or BinLorry role, whereas, the Field component can take one of three possible roles: DirtySpot, CleanSpot, and Bin role. Each of these roles has a state (data) and programmed behaviors. Further, these components are coordinated as three dynamic SeSaMe groups: (i) the Cleaning group which comprises components assigned to the cleaning task, (ii) the Disposal group consisting of components participating in the disposal activity, and the (iii) Room grouping all Fields of a room.

In SeSaMe, a group is comprised of a component with a *supervisor* role and various components with *follower* roles. Therefore, in the current context, a robot can play both the supervisor and follower roles (as a Cleaner or BinLorry) with respect to its participation in various groups (Cleaning or Disposal). On the other hand, a Room group may be composed with certain Cleaning group(s) within a proximity. SeSaMe creates these dynamic groups at runtime, based on the defined rules (grouping policies) [15] and, hence, allows for the formation of ad-hoc collaborations to facilitate role-oriented knowledge exchange among member components of a group. In this way, SeSaMe enables various subsystems to take part in multiple collaborations (groups) and with each collaboration their knowledge is extended. This strategy improves performance and memory consumption but potentially results in decreasing the optimality of local decision-making as decision making is done at group-level.

Although SeSaMe has the notion of roles, it does not inherently support dynamic and adaptive knowledge exchange. We believe that an approach that can make use of the adaptive knowledge exchange and implements the third strategy, can significantly improve the performance and memory consumption of highly dynamic systems.

[1]Here, gossiping takes the form of probabilistic broadcasting.

VII. Conclusion

In this paper we presented a *role-based adaptive knowledge exchange technique* for *partial runtime models*. Our approach supports adaptation w.r.t. the runtime type of knowledge and conditions over it. We showed three strategies for knowledge exchange between collaborating subsystems, introduced role-based runtime models for partial views and illustrated their use for role-based knowledge exchange. Our technique enables cooperating subsystems to reduce the amount of knowledge exchanged, which consequently helps to improve the overall performance, to lower energy and memory consumption, as well as to reduce privacy threats. Moreover, our approach lowers maintenance efforts for queries between subsystems by capturing collaboration-specific knowledge in role types. We discussed the general technique and two possible realizations using the DEECo and SeSaMe technologies.

As future work, we plan to build a general framework capturing the protocol of interaction for highly dynamic systems as described in the paper and will show the optimality of the third strategy. For this, we will do prototypical implementations in DEECo and SeSaME. Next, we will investigate the correlation of the amount of knowledge K with the corresponding utility $U(K)$ and cost $C(K)$ by comparing the best possible decisions a subsystem made with complete knowledge with those it made with partial knowledge.

Acknowledgment

The authors would like to thank the GI-Dagstuhl Seminar 14433 - Software Engineering for Self-Adaptive Systems that stimulated initial discussions about this work. This work is partially financed by the Knowledge Foundation through the Internet of Things and People research profile (Malmö University, Sweden), by the German Research Foundation within the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing", by the European Union Seventh Framework Programme FP7-PEOPLE-2010-ITN under grant agreement n°264840, by the Joint Open Lab "S-Cube" - Telecom Italia S.p.A. Innovation division, Italy, and by the Datalyse project www.datalyse.fr.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, ser. ISORC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 363–369. [Online]. Available: http://dx.doi.org/10.1109/ISORC.2008.25

[2] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE (special issue on CPS)*, vol. 100, no. 1, pp. 13 – 28, January 2012. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/843.html

[3] B. K.-d. Kim and P. R. Kumar, "CyberPhysical Systems: A Perspective at the Centennial," *Proceedings of the IEEE*, vol. 100, no. Special Centennial, pp. 1287–1308, 2012.

[4] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Goeschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, M. Litoiu, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, B. Schmerl, D. B. Smith, J. P. Sousa, G. Tamura, L. Tahvildari, N. M. Villegas, T. Vogel, D. Weyns, K. Wong, and J. Wuttke, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds. Springer-Verlag, 2013, vol. 7475, pp. 1–32.

[5] J. Keznikl, T. Bures, F. Plasil, I. Gerostathopoulos, P. Hnetynka, and N. Hoch, "Design of ensemble-based component systems by invariant refinement," in *Proc. of CBSE'13*. ACM, Jun. 2013, pp. 91–100.

[6] G. Blair, N. Bencomo, and R. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, Oct 2009.

[7] U. Aßmann, S. Götz, J.-M. Jézéquel, B. Morin, and M. Trapp, *Models@run.time*. Springer LNCS, 2014, vol. 8378, ch. A Reference Architecture and Roadmap for Models@run.time Systems.

[8] A. C. Kay, "The early history of smalltalk," in *HOPL Preprints*, 1993, pp. 69–95.

[9] C. W. Bachman, "The programmer as navigator," *Commun. ACM*, vol. 16, no. 11, pp. 635–658, 1973.

[10] T. Kühn, M. Leuthäuser, S. Götz, C. Seidl, and U. Aßmann, "A metamodel family for role-based modeling and programming languages," in *Software Language Engineering*, ser. Lecture Notes in Computer Science, B. Combemale, D. Pearce, O. Barais, and J. Vinju, Eds. Springer International Publishing, 2014, vol. 8706, pp. 141–160.

[11] N. Guarino and C. Welty, "A formal ontology of properties," in *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*. Springer, 2000, pp. 97–112.

[12] D. Riehle and T. Gross, "Role model based framework design and integration," in *Proceedings of the 1998 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '98)*. ACM Press, 1998, pp. 117–133.

[13] T. Jäkel, T. Kühn, H. Voigt, and W. Lehner, "Rsql - a query language for dynamic data types," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, ser. IDEAS '14. New York, NY, USA: ACM, 2014, pp. 185–194. [Online]. Available: http://doi.acm.org/10.1145/2628194.2628246

[14] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "DEECo: An Ensemble-based Component System," in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ser. CBSE '13. New York, NY, USA: ACM, 2013, pp. 81–90. [Online]. Available: http://doi.acm.org/10.1145/2465449.2465462

[15] L. Baresi, S. Guinea, and A. Shahzada, "SeSaMe: Towards a Semantic Self Adaptive Middleware for Smart Spaces," in *Post-Proceeding of The Workshop on Engineering Multi-Agent Systems*, 2013, pp. 169–178.

[16] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, and A. Koukam, "Aspecs: an agent-oriented software process for engineering complex systems," *In Autonomous Agents and Multi-Agent Systems*, vol. 20, no. 2, pp. 260–304, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10458-009-9099-4

[17] D. Cassou, J. Bruneau, C. Consel, and E. Balland, "Toward a Tool-Based Development Methodology for Pervasive Computing Applications," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1445–1463, Nov 2012.

[18] G. Castelli, M. Mamei, A. Rosi, and F. Zambonelli, "Pervasive middleware goes social: The sapere approach," in *Proceeding of 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops, Michigan, USA*, 2011, pp. 9–14.

[19] R. Spalazzese, P. Inverardi, and V. Issarny, "Towards a formalization of mediating connectors for on the fly interoperability," in *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA 2009)*, 2009, pp. 345–348.

[20] P. Inverardi, V. Issarny, and R. Spalazzese, "A theory of mediators for eternal connectors," in *Proceedings of ISoLA 2010 - 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Part II.*, vol. 6416. Springer, Heidelberg, 2010, pp. 236–250.

[21] A. Fipa, "Fipa acl message structure specification," *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*, 2002.

[22] N. Bencomo, A. Bennaceur, P. Grace, G. Blair, and V. Issarny, "The role of models@ run. time in supporting on-the-fly interoperability," *Computing*, vol. 95, no. 3, pp. 167–190, 2013.

[23] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "Gossiping Components for Cyber-Physical Systems," in *Proc. of 8th European Conference on Software Architecture*. Springer, 2014, pp. 250–266.