# An Actuation Conflicts Management Flow For Smart IoT-based Systems

Gérald Rocher*, Thibaut Gonnin*, Franck Dechavanne*, Stéphane Lavirotte*, Jean-Yves Tigli*,
Laurent Capocchi† and Jean-François Santucci†
*Laboratoire I3S
Université Côte d'Azur, CNRS, Sophia-Antipolis, France
Email: firstname.lastname@univ-cotedazur.fr
†SPE UMR CNRS 6134
Université de Corse, Corte, France
Email: {capocchi,santucci}@univ-corse.fr

***Abstract*—IoT-based applications have long been limited to collecting field information; at the edge of their underlying infrastructure, IoT devices utilization is mainly motivated by their capacity at gathering environmental information from sensors as means to support users in decision making. However, in numerous domains like home automation, smart factory, intelligent transportation systems, etc., so-called 'smart' IoT-based applications also involve devices interacting with the physical environment through actuators. Throughout their life cycle, from the design, the deployment to operation, the ability to prevent conflicting actuation commands and antagonistic effects (possibly harmful), represents a new challenge in the realm of trustworthy smart IoT-based applications. In this paper, we introduce a complete flow for identifying and resolving actuation conflicts at design time. The proposed approach is part of the DevOps software development life cycle. It advocates the reuse of conflict management solutions through local resolution strategies, considering asynchronous timings of targeted hardware platforms they are deployed on. An illustration of the flow is provided on a use-case.**

***Index Terms*—IoT, actuation conflicts, features interaction, model driven engineering, trustworthiness, Discrete Event Specification Formalism, DevOps**

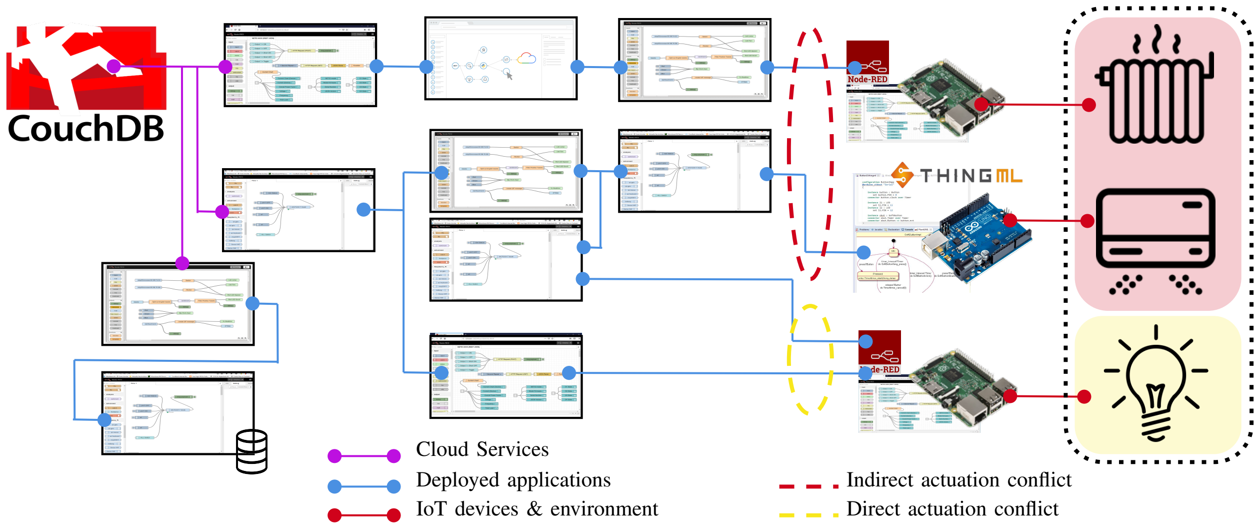
## I. Introduction

For a long time, IoT devices utilization at the edge of IoT-based systems infrastructures, has been motivated by their capacity at collecting environmental information from sensors, paving the way for decision making support systems covering a wide range of application domains from smart-health, smart-city to smart-grid, etc. just to name a few. IoT devices being often *shared* between applications, multi-layered architectures thus appeared, commonly broken down into a three-layered architecture : (1) a shared infrastructure layer, (2) a top layer where applications are deployed and (3) an intermediate layer ensuring the overall consistency between applications at top level and the shared infrastructure at the bottom level. The challenge addressed is then rather technological and aims to provide the shared infrastructure layer with sensors access control mechanisms [1].

However, new challenges appear as soon as it comes to smart applications to control shared IoT-devices embedding actuators that turn commands into physical effects. One of these challenges is the management of *actuation conflicts* that may arise as soon as different applications compete for accessing shared actuators (*direct conflicts*) or shared physical properties (*indirect conflicts*). For instance, let us consider a smart-home scenario (fig.1); a first application controlling lights and blinds for energy consumption reduction purpose, compete with a second application contributing to users well-being by controlling the same lights and blinds and any other actuators relevant for this purpose. At any time, both applications are likely to trigger *antagonistic commands* to these shared actuators leading a direct conflict to occur. Relying on the previous scenario, one can also consider both applications to control home temperature through heaters, coolers, blinds, etc. In this context, an indirect conflict is likely to occur when both applications trigger commands to the heater and the cooler, i.e. *to different actuators competing to the same physical property*, the temperature in that case. Relying on actuators to achieve their purposes, IoT-based applications are no longer isolated processes immune to physical effects induced by concurrent applications operating in shared environments [2].

Along with the technological challenge, actuation conflict thus induces a *semantical challenge*; taking into account the locality of the actuators and the physical properties they interact with, is here essential, characterized by the concept of “entity of interest” [3], [4]. This challenge cannot be ignored nor be delegated to end-users [5] and aims to provide the shared infrastructure layer with Actuation Conflict Management (ACM) mechanisms. In the realm of trustworthy smart IoT-based applications, conflict management is of paramount importance and requires decision support tools that can assist designers in identifying and resolving direct and indirect conflicts, and in deploying relevant, yet robust and safe ACMs [6].

In this paper, a complete flow for identifying and locally resolving direct and indirect actuation conflicts in the realm of trustworthy IoT-based applications is introduced. This research is part of the ENACT European project [7] which aims to provide complete DevOps support for trustworthy Smart IoT Systems (SIS). The contribution is threefold:



CouchDB
Node-RED
THINGML
Cloud Services
Deployed applications
IoT devices & environment
Indirect actuation conflict
Direct actuation conflict

Fig. 1: Applications deployment and underlying direct/indirect conflicts at the edge of the infrastructure

1) Underlying the tools for actuation conflicts identification and resolution, a metamodel, denoted WIMAC (Workflow and Interaction Model for Actuation Conflict management), is presented. On the basis of GeneSIS [8] deployment model, implementation models and a model of the physical environment, it provides a language for modelling inter-relationships between software components and actuators at the edge of the infrastructure layer along with their effects on the physical environment. The proposed approach does not require an a priori knowledge on applications internal logic,
2) DevOps approach aims to provide continuous and rapid software deployment capabilities. In line with this approach, a set of pre-configured off-the-shelf and ready-to-use ACMs are offered to designers, allowing local resolution of conflicts identified in the design,
3) A complete formal verification flow is proposed for designing reusable custom ACMs. Platforms at the edge of the infrastructure are highly heterogeneous, resource-constrained (communication delays, etc.) and likely governed by asynchronous events [9]. In this context, besides logical properties, temporal properties verification's is proposed in Discrete EVent system Specification Formalism (DEVS [10]), allowing to validate ACMs behaviour for different implementation strategies. Moreover, DEVS offers a common representation for different discrete event modelling frameworks ACMs can be based on [11] and, in end-of-pipe, allows to build a library of reusable DEVS-based ACMs (a.k.a., DEVS kernels) targeting the different implementation strategies (i.e., hardware platforms)

## II. Related work

The problem of identifying and resolving actuation conflicts in the context of IoT-based applications is relatively new. Thus, the scientific community's efforts to address this problem are still in their infancy and, while most existing solutions focus on identifying conflicts, few focus on proposing their resolution [12].

In [6] authors present a watchdog architecture for direct and indirect actuation conflicts identification and resolution. All actions requested by services involved are intercepted and analysed for conflicts. In case of conflicts, their resolution is achieved from (1) an Optimization-oriented Decision Making (ODM) module using some pre-defined policies and (2), a Preference Learning (PL) module, learning preferences from users and further used in ODM module. In [13], authors address both direct and indirect actuation conflicts identification and resolution for large-scale IoT systems using formal methods. The approach relies on policies whose violation results in conflicts detection. Actuation conflicts identification and resolution are achieved through a centralized controller that requires, among others, the functional logic of the software components involved to be known. In [12] and close to [6], authors propose an approach to detect direct and indirect conflicts in a given set of distributed IoT applications with respect to a set of rules that define the allowable and restricted state-space transitions of devices. The proposed approach requires all events to be intercepted for further analysis. For each identified conflict, a set of remedial actions are suggested to the user. Both identification and resolution mechanisms are handled by centralized modules (Conflict Detector and Remedial Action engine).

Although some of the above solutions propose conflicts resolution at runtime (e.g., [6]), most of them require an a priori knowledge on the components of the system considered and the rules governing their evolution [14]. They do not propose local and reusable solutions to conflicts. On the contrary, they implement global identification and resolution mechanisms that are not easily reusable. Finally, no approach considers the heterogeneity of software platforms at the edge of the infrastructure and their temporal specificities characterized by the asynchronism of the events around them. tackle large number or

In [15], it is recognized that interactions between devices are an increasing cause of safety and security violations whose detection "requires a holistic view of installed apps, component devices, their configurations, and more importantly, how they interact". This is the approach followed in this article whose details are presented in the sequel.

## III. Overall approach

IoT-based applications often implement and devices that, through actuators and sensors, have an immediate impact for users who use them daily. Thus, facilitating the maintainability of these applications, i.e. deploying in a transparent, rapid and automatic way patches and updates both at the level of the applications and the devices they implement, is all the more important. In this context, the DevOps development approach is a competitive differentiator, broadly applied in the IoT world [16]. This approach aims to provide continuous and rapid software deployment capabilities thanks to a set of tools and models shared across stakeholders engaged in the process.

In the context of DevOps, tools for identifying and resolving direct and indirect actuation conflicts can rely on *deployment and implementation models* from which interactions of deployed applications with the underlying infrastructure, down to the devices and the sensors and/or actuators they embed, can be extracted. So as to identify indirect actuation conflicts, actuators description, i.e., the effects they produce and the environment they operate in, has to be provided. This model, denoted by *physical environment model* in the sequel, is generally not part of the deployment and implementation models. Unless semantic annotations are added in deployment and/or implementation models, it has to be provided by designers at design time.

On the basis of these models, a global description model is generated, relying on a metamodel denoted WIMAC (Workflow and Interaction Model for Actuation Conflict management). This metamodel (fig.2) provides a language for describing inter-relationships between software components and sensors/actuators at the edge of the infrastructure layer along with their effects on the physical environment. WIMAC metamodel main entities are the following:

- **SoftwareComponents** are black-box components. In this context, an application could be a single SoftwareComponent or a composition of SoftwareComponents described through an implementation model (e.g., Node-Red flow). It is also worth noting that ACMs only rely on input/output for infering software components inter-relationships, identifying and resolving conflicts hereby, keeping unchanged the software components logic,
- **ActionComponents** are SoftwareComponents controlling *transducers* (i.e., sensors and/or actuators),
- **PhysicalSystems** are physical entities bounded in space whose some of their physical properties evolution (e.g. temperature in the kitchen), are driven by ActionComponents.

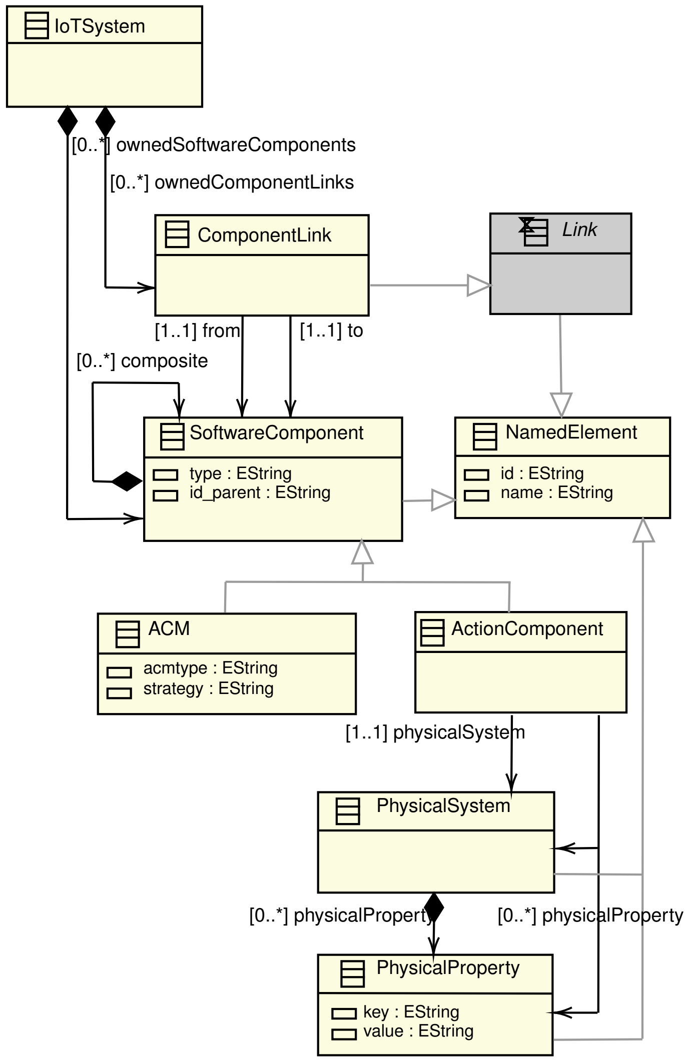


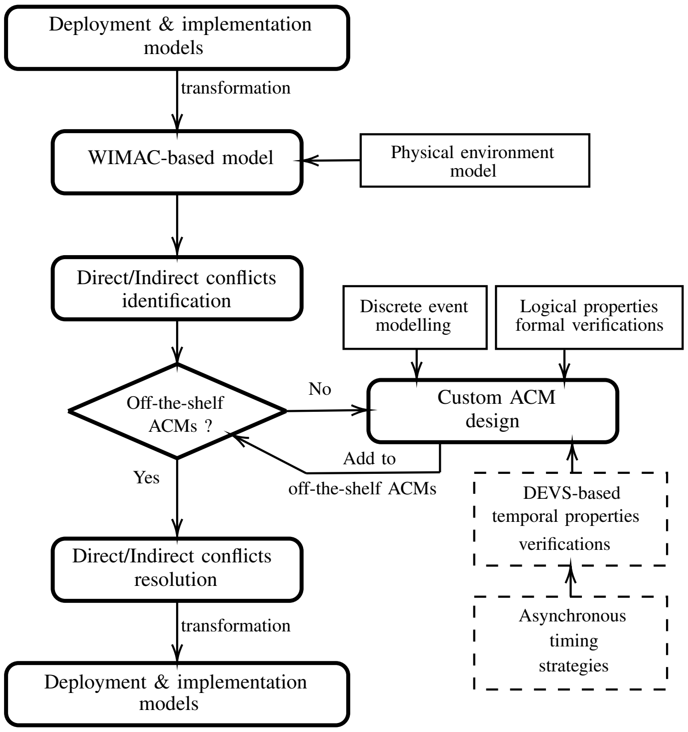
Fig. 2: WIMAC metamodel

The whole approach proposed in this paper for identifying and resolving IoT-based applications actuation conflicts is depicted in fig.3. A first tool extracts the WIMAC-based model from the deployment, implementation and the physical environment models. In the context of this paper, the deployment model relies on GeneSIS modelling language [8], providing WIMAC extraction tool with necessary information on the components to be deployed. The WIMAC-based model can then be used to automatically identify points of direct and indirect actuation conflict. From that point, conflicts resolution can be achieved at two levels:

- At the first level, predefined conflict patterns are automatically identified. Off-the-shelf and ready-to-use ACM solutions are then proposed to designers to solve them,
- At the second level (optional), designers are provided with tools for designing custom, yet robust and safe ACMs solutions: (1) Conceptual models are specified in the form of discrete event Finite State Machines (FSM), their logical properties can be verified using state of the art formal methods; (2) Acknowledging the fact that

platforms at the edge of the infrastructure are highly heterogeneous and subject to asynchronous behaviours, implementation models are specified in the DEVS formalism [10] allowing to conduct simulations and temporal properties verifications for different targeted platforms and, in end-of-pipe, to generate their associated software components. Once designed and verified, custom ACMs can be added in off-the-shelf ACMs.

Finally, concrete ACMs are instantiated, and the WIMAC-based model is transformed back to a GENESIS-based deployment model; any subsequent design modifications trigger a new conflicts resolution sequence. In the sequel, the two aforementioned levels are detailed, i.e., actuation conflicts identification and resolution using off-the-shelf ready-to-use ACMs and the design of custom ACMs.



Deployment & implementation models
transformation
WIMAC-based model
Physical environment model
Direct/Indirect conflicts identification
Discrete event modelling
Logical properties formal verifications
Off-the-shelf ACMs ?
No
Custom ACM design
Add to off-the-shelf ACMs
Yes
DEVS-based temporal properties verifications
Direct/Indirect conflicts resolution
Asynchronous timing strategies
transformation
Deployment & implementation models

Fig. 3: Actuation conflicts identification and resolution flow

## IV. OFF-THE-SHELF ACM SOLUTIONS

DevOps approach aims to provide continuous and rapid software deployment capabilities. To support this objective, a tool has been developed relying on a common WIMAC-based model to which graph and model transformation algorithms are applied. The purpose of this tool is to automatically identify conflict patterns and apply graph transformations to instantiate off-the-shelf ready-to-use ACMs. Conflicts identification and transformation are defined by a set of predefined *Attributed Graph Grammar* (AGG) rules [17] in the form of attributed graphs. Algebraic graph rewriting via a Double-PushOut (DPO) [18] approach is used for applying graph transformations. In order for the tool to apply AGG-based patterns search and graph transformation rules, the WIMAC-based model has to be itself transformed into an AGG model (fig.6). Examples of direct/indirect actuation conflict patterns and their associated graph transformation rules are respectively depicted in fig.4 and fig.5.

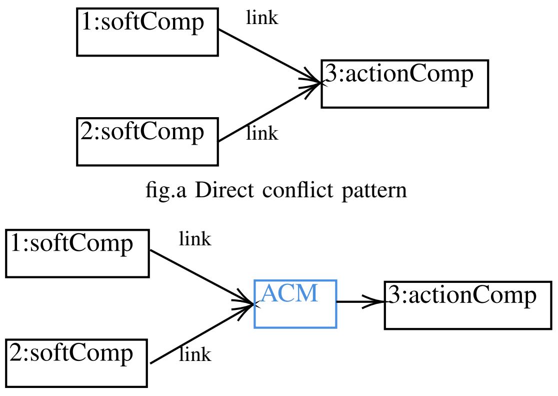


fig.b Graph transformation rule

Fig. 4: Graph rewriting rule for direct conflict

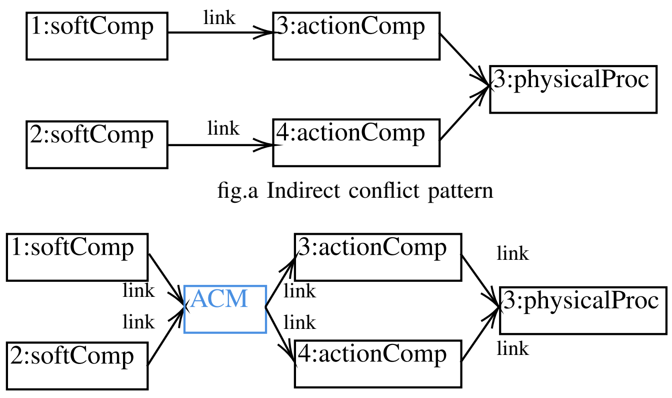


fig.b Graph transformation rule

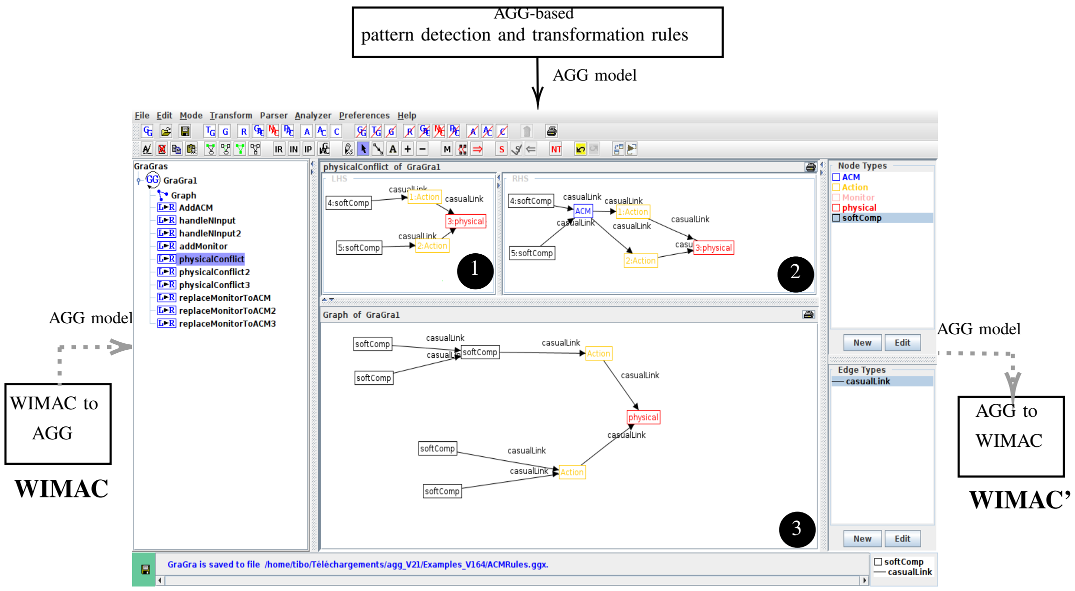
Fig. 5: Graph rewriting rule for indirect conflict

Once actuation conflicts have been identified, a set of off-the-shelf ready-to-use ACMs are proposed to designers. Each ACM comes pre-compiled for a set of predefined deployment target platforms. This approach requires designers of the incriminated software components to find a trade-off on the ACM to be instantiated before applications are deployed. Such a collaborative work methodology is at the heart of the DevOps approach.

In some particular cases, however, off-the-shelf ACMs may not fit the needs, leading designers to develop custom ACMs.

## V. CUSTOM ACMS DESIGN

While off-the-shelf ACMs instantiation are helpful for resolving common actuation conflicts patterns, it may be still necessary for designers to develop custom, yet robust and safe ACMs to address specific actuation conflicts. To this end, Model Driven Engineering (MDE) tools are leveraged allowing two implementation levels (fig.7):



GraGra is saved to file /home/tibo/Téléchargements/agg_V21/Examples_V164/ACMRules.ggx.

Fig. 6: AGG-based actuation conflicts identification. ❶ corresponds to the AGG conflict pattern to be identified, ❷ corresponds to the graph transformation rule (where an ACM is instantiated) and ❸ corresponds to the WIMAC subgraph where the conflict is identified.

- At a first level, *conceptual models* allow custom ACMs *logical properties* (e.g. completeness, safety, liveness, etc.) to be formally verified by using state of the art methodologies. In this paper, custom ACMs are thus defined through Finite State Machines (FSM),
- At a second level, *implementation models* allow custom ACMs *temporal properties* to be formally verified through different *asynchronous execution machine strategies*. In this paper, DEVS (Discrete EVent system Specification) [10] formalism is leveraged for simulating the different implementation strategies. It is worth noting that it has been shown in [19] that any discrete event behaviour can be expressed as a DEVS model.

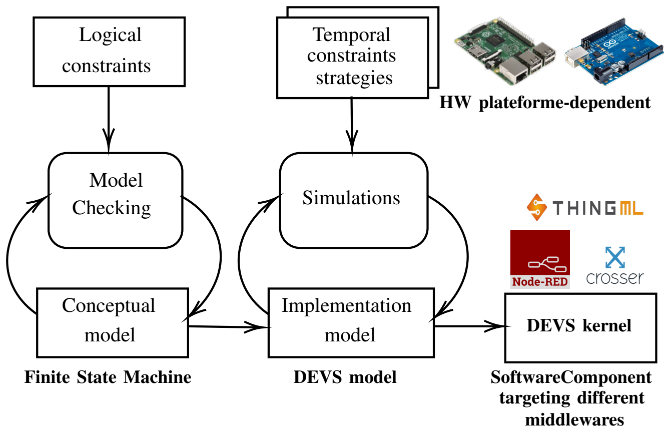


Fig. 7: Custom ACM design process

As part of the design of custom ACMs, DEVS formalism has the following key advantages:

- It enables the encapsulation of synchronous discrete event models into asynchronous environments. This is of paramount importance when considering that software components are likely to be deployed on different *resource-constrained hardware platforms* at the edge of the infrastructure, subject to timing constraints relative to wireless communication protocols, computational capabilities, etc., leading asynchronous behaviours to likely appear [9],
- It provides a common representation for different existing discrete event modelling formalisms (including Petri Nets, FSM, and different state machines)[11]. Thus, designers are not limited to a particular modelling framework when designing custom ACMs,
- It allows to build a library of reusable DEVS-based ACMs (a.k.a., DEVS kernels) targeting different implementation strategies (i.e., hardware platforms).

In this paper, DEVS-based ACMs are modelled through DEVS *atomic models* defined by (fig.8)

DEVSAtomic $=< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$ where

- $X = \{(p, v)|p \in \text{InputPorts}, v \in X_p\}$ where InputPorts is the set of input ports and $X_p$ is the set of possible values for these ports,
- $Y = \{(p, v)|p \in \text{OutputPorts}, v \in Y_p\}$ where OutputPorts is the set of output ports and $Y_p$ is the set of possible values for these ports,
- $S$ is the set of system states,
- $ta : S \to \mathbb{R}_+$ defines the lifespan for each state. When $ta(s) = \infty$ the lifespan of the state $s \in S$ is unconstrained whereas $ta(s) = 0$ leads an immediate transition to the next state,

- $\delta_{int} : S \rightarrow S$ is the *internal* state transition function,
- $\delta_{ext} : Q \times X \rightarrow S$ is the *external* state transition function with $Q = \{(s, e) | s \in S, e \in [0, ta(s)]\}$,
- $\lambda : S \rightarrow Y$ is the output function.

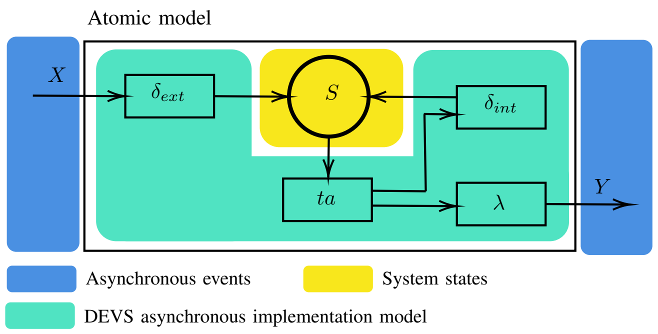


Fig. 8: DEVS atomic model

The behaviour of an atomic model can be understood as follows [20]. Let us Consider that the system is in the state $s \in S$ at time $t$. Unless an external event occurs on one of the input ports $X$, the system remains in state $s$ for a time $d$ defined by $d = ta(s)$. When $ta(s)$ expires, the model sends, on one of the output ports $Y_p$, the value given by $\lambda(s)$ and evolves to a new state $s \in S$ defined by $\delta_{int}(s)$. If an external event $x \in X$ occurs on one of the input ports $X_p$, before the expiration of $d = ta(s)$, it triggers an external transition. In this case, $\delta_{ext}(s, t_e, x)$ defines which state is the next state $s'$ (where $s$ is the current state, $t_e$ is the time elapsed since the last transition, and $x \in X$ is the received event).

The objective is then to define a set of asynchronous timings management strategies for targeted hardware platforms. Several modeling and simulation tools have been developed to facilitate this process [21], [22]. In the context of this paper, DEVSimPy [20] tool is used.

Following this approach, a set of reusable strategies can be developed and maintained, targeting different deployment hardware platforms while keeping unchanged the conceptual models. The final solution can then be compiled as C++, C# or any high-level programming language for further deployment on middleware/EDGE solutions (node-red, crosser I/O, etc.). Designing robust and safe custom ACMs may be quite complex but it is worth noting that once designed, it can be integrated as off-the-shelf solutions and reused as needed.

## VI. Illustration

To illustrate the flow described throughout this paper, let us consider a simple use-case in the smart-office domain. In a first step, two independent offices whose access is granted either through a RFID card reader or a button are considered (fig.9). In this example, two actuators are involved (door locking system). The software management of these elements is implemented on two Arduino boards attached to a Raspberry Pi board (gateway).

On the basis of this model and the associated deployment and implementation models, a WIMAC-based model

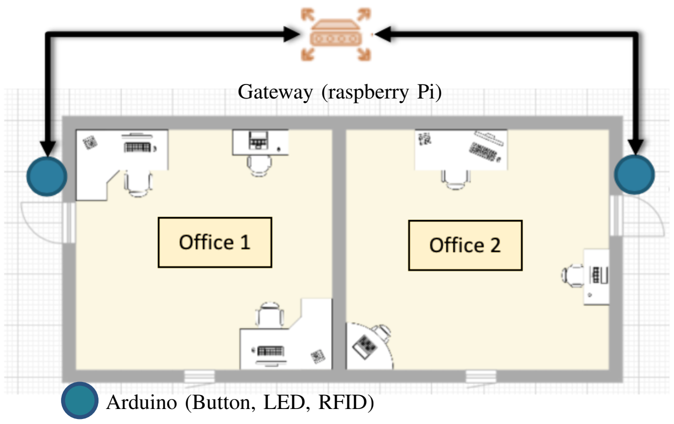
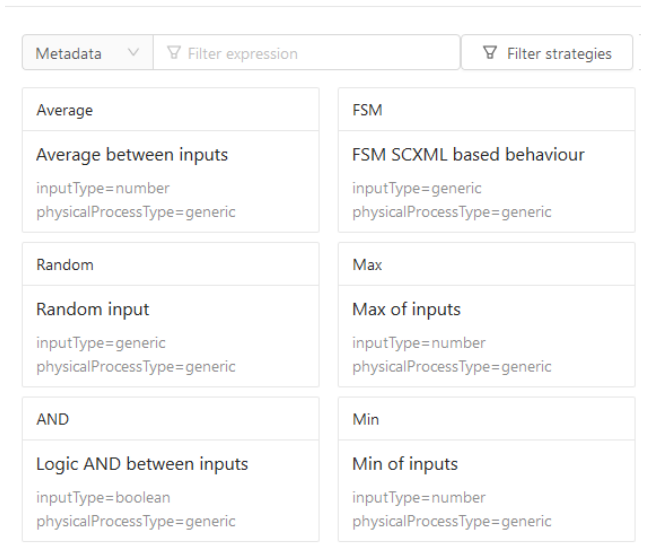


Fig. 9: Two independent controlled access offices

is generated, modelling inter-relationships between software components and actuators. For instance, the WIMAC-based model associated to the illustration depicted in fig.9 is depicted in fig.11. It is worth noting that the physical environment model can be defined here by designers. It appears through $\varphi$ symbol in fig.11 and fig.12. In the context of this first illustration, both actuators are associated with each office.

Given the WIMAC model, the actuation conflict identification and resolution tools are executed. Two direct conflicts are identified on the locks (indeed, doors can either be unlocked by RFID card readers or buttons). ACMs instances are instantiated at the conflict points as depicted in fig.11. In this figure, one conflict has been fixed by the designer, one remains to be fixed. At that point, a set of off-the-shelf ready-to-use concrete ACMs are proposed to designers who can select the most appropriate given the situation (fig.10).



Average
Average between inputs
inputType=number
physicalProcessType=generic

FSM
FSM SCXML based behaviour
inputType=generic
physicalProcessType=generic

Random
Random input
inputType=generic
physicalProcessType=generic

Max
Max of inputs
inputType=number
physicalProcessType=generic

AND
Logic AND between inputs
inputType=boolean
physicalProcessType=generic

Min
Min of inputs
inputType=number
physicalProcessType=generic

Fig. 10: Off-the-shelf ready-to-use ACM solutions.

Now, let us consider that the configuration is changed such that offices are merged into a meeting room (fig.13). Actually, the deployment model remains identical, only the physical environment model is modified (fig.12) to reflect the fact that,

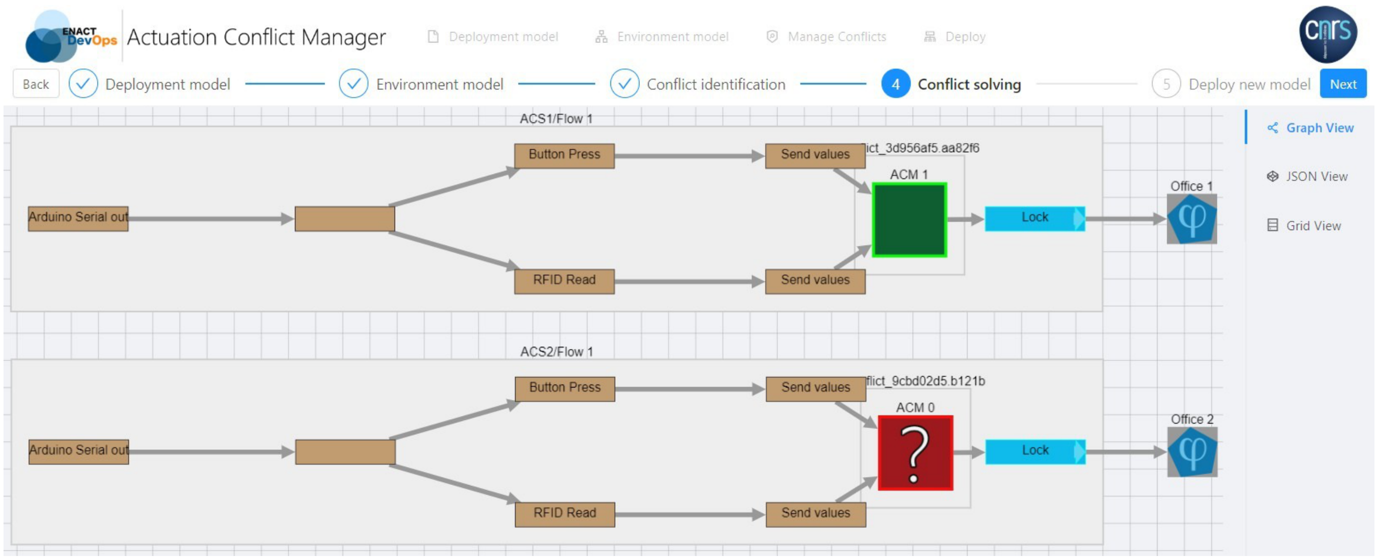


Fig. 11: Two direct conflicts are identified, one has been fixed by designer, one remains to be fixed.

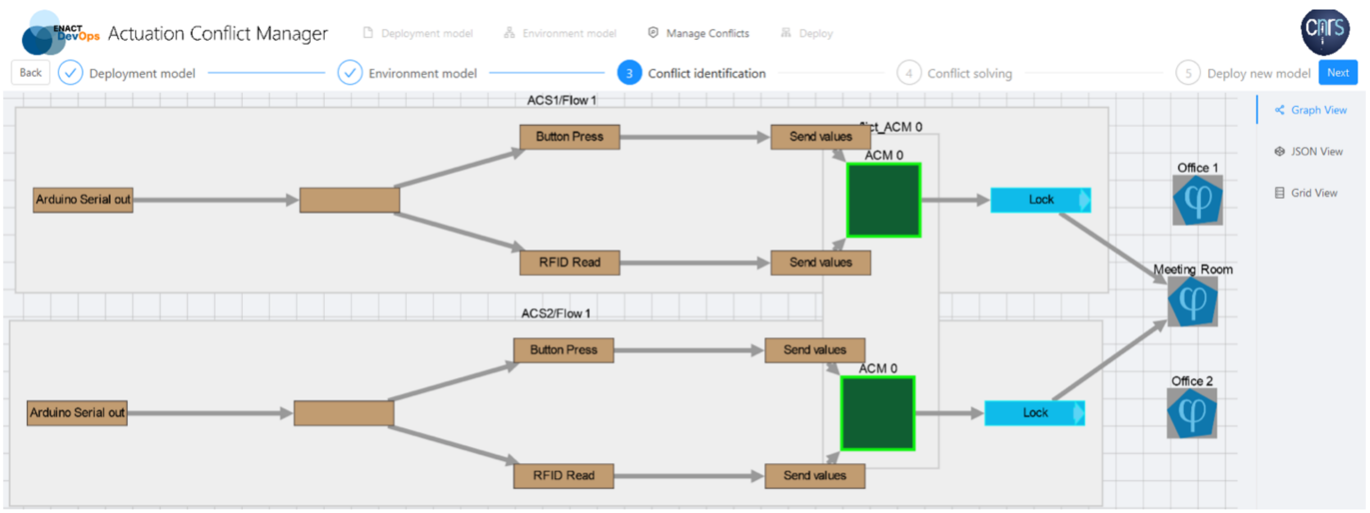


Fig. 12: An indirect conflict is identified in addition with the two previously identified direct conflicts.

under this new configuration, both actuators (door locking systems) impact the same physical environment (the meeting room).

In this context, besides direct conflicts previously identified, an indirect conflict is also identified. In this particular case, this conflict is expected and does not lend to consequence in the context of multiple access to a room. Thus, the indirect conflict manager can be replaced with a simple passthrough ACM.

## VII. Conclusion and Perspectives

In the realm of trustworthy IoT-based applications, actuators implementation requires new development tools to help designers identify and resolve, as early as possible, the direct and indirect actuation conflicts that may occur and lead unexpected (potentially harmful) behaviours. In this paper, a complete flow, taking part of the DevOps software development life cy-

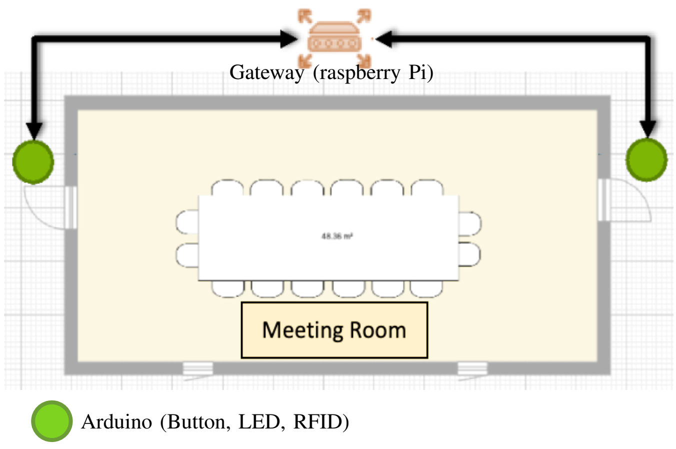


Fig. 13: Both offices are merged into a meeting room

cle, has been presented for identifying and resolving actuation conflicts at design time. Unlike most existing approaches based on centralized solutions, it advocates the reuse of conflict management solutions through local resolution strategies.

The proposed flow facilitates and accelerates IoT-based software development through a set of off-the-shelf ready-to-use conflicts management solutions targeting (1) different hardware platforms with different asynchronous timing specifications thanks to the Discrete EVent system Specification Formalism (DEVS) validations and (2) different middleware (node-red, crosser io, ThingML, etc.) taking into account devices heterogeneity at the edge of the infrastructure. Opportunity is given designers to develop reusable custom ACMs from conceptual models (Finite State Machine (FSM) have been used in the context of this paper but any discrete event based formalism can be used depending on designers preferences) whose logical constraints can be verified through state of the art formal verification techniques, comprehended with DEVS-based implementation models for asynchronous timings strategies verifications.

Some investigations and improvements are envisioned. The proposed flow has been validated in the Smart-office domain, involving a dozen applications and actuators. We do plan to stress to proposed flow in the smart-city domain which involves much more applications and actuators. For the time being, off-the-shelf ACMs are pre-synthesized for a set of platforms. We do plan to have it synthetized on the fly depending on the targeted platform specified on the deployment and implementation models thus allowing to only record conceptual models. A second improvement envisioned consists in leveraging web semantics technologies for inferring direct and indirect conflicts and proposing relevant ACMs on the basis of semantic annotations added into the deployment and implementation models.

## Acknowledgment

The research leading to these results has received funding from the European Commission's H2020 Program under grant agreement numbers 780351 (ENACT).

## References

[1] C. Cecchinel, S. Mosser, and P. Collet, "Software development support for shared sensing infrastructures: A generative and dynamic approach," in *International Conference on Software Reuse*. Springer, 2015, pp. 221–236.

[2] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, "Service oriented middleware for the internet of things: a perspective," in *European Conference on a Service-Based Internet*. Springer, 2011, pp. 220–229.

[3] S. Haller, "The things in the internet of things," *Poster at the (IoT 2010). Tokyo, Japan, November*, vol. 5, no. 8, pp. 26–30, 2010.

[4] I. Sarray, A. Ressouche, D. Gaffé, J.-Y. Tigli, and S. Lavirotte, "Safe composition in middleware for the internet of things," in *Proceedings of the 2nd Workshop on Middleware for Context-Aware Applications in the IoT*, 2015, pp. 7–12.

[5] J. Song, A. Kunz, M. Schmidt, and P. Szczytowski, "Connecting and managing m2m devices in the future internet," *Mobile Networks and Applications*, vol. 19, no. 1, pp. 4–17, 2014.

[6] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic, "Detection of runtime conflicts among services in smart cities," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2016, pp. 1–10.

[7] N. Ferry, A. Solberg, H. Song, S. Lavirotte, J.-Y. Tigli, T. Winter, V. Muntés-Mulero, A. Metzger, E. R. Velasco, and A. C. Aguirre, "Enact: Development, operation, and quality assurance of trustworthy smart iot systems," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers*, vol. 11350. Springer, 2018, p. 112.

[8] N. Ferry, P. Nguyen, H. Song, P.-E. Novac, S. Lavirotte, J.-Y. Tigli, and A. Solberg, "Genesis: Continuous orchestration and deployment of smart iot systems," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2019, pp. 870–875.

[9] A. J. Van Der Schaft and J. M. Schumacher, *An introduction to hybrid dynamical systems*. Springer London, 2000, vol. 251.

[10] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018.

[11] T. Zheng and G. A. Wainer, "Implementing finite state machines using the cd++ toolkit," in *In Proceedings of the SCS Summer Computer Simulation Conference, 2003. atomic model, 1 CD++, 1 coupled model, 1 DEVS, 1 DEVS Graph, 1 discrete-event modeling*. Citeseer, 2003.

[12] R. Liu, Z. Wang, L. Garcia, and M. Srivastava, "Remediot: Remedial actions for internet-of-things conflicts," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2019, pp. 101–110.

[13] A. Al Farooq, E. Al-Shaer, T. Moyer, and K. Kant, "Iotc 2: A formal method approach for detecting conflicts in large scale iot systems," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 442–447.

[14] T. Shah, S. Venkatesan, T. Ngo, K. Neelamegam *et al.*, "Conflict detection in rule based iot systems," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2019, pp. 0276–0284.

[15] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "Iotsan: Fortifying the safety of iot systems," in *Proc. of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 191–203.

[16] F. Ahmadighohandizi and K. Systä, "Application development and deployment for iot devices," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2016, pp. 74–85.

[17] C. Jones, "Attributed graphs, graph-grammars, and structured modeling," *Annals of Operations Research*, vol. 38, no. 1, pp. 281–324, 1992.

[18] B. König, D. Nolte, J. Padberg, and A. Rensink, "A tutorial on graph transformation," in *Graph Transformation, Specifications, and Nets*. Springer, 2018, pp. 83–104.

[19] K. T. Gon, B. P. Zeigler, and H. Praehofer, "Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems," 2000.

[20] L. Capocchi, J. F. Santucci, B. Poggi, and C. Nicolai, "Devsimpy: A collaborative python software for modeling and simulation of devs systems," in *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2011, pp. 170–175.

[21] D. Niyonkuru and G. Wainer, "Towards a devs-based operating system," in *Proc. of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 101–112. [Online]. Available: https://doi.org/10.1145/2769458.2769465

[22] Y. Van Tendeloo and H. Vangheluwe, "An evaluation of devs simulation tools," *Simulation*, vol. 93, no. 2, pp. 103–121, 2017.