A Hybrid Modelling Approach for Hierarchical Control of Structured CPSs ⋆

Simon Bliudze[1][0000−0002−7900−5271], Sophie Cerf[1][0000−0003−0122−0796], and Olga Kouchnarenko[2][0000−0003−1482−9015] ⋆⋆

[1] Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[2] Université Marie et Louis Pasteur, CNRS UMR6174, Institut FEMTO-ST, F-25000 Besançon, France FirstName.LastName@inria.fr[1],Olga.Kouchnarenko@femto-st.fr[2]

Abstract. Cyber-physical systems (CPSs) include engineered interacting networks of physical and computational components. As they are widely used in many application domains, guaranteeing their correct and proper behaviour is an essential and a challenging issue. This paper aims to contribute to a flexible design and development of *structured* CPSs, composed of similar elements, and capable of (self-)adaptation to satisfy evolving internal and external constraints, e.g. using control theory. To this end, we make use of their structure and of their behavioural characteristics for modelling by hierarchical motifs both systems' elements and controllers. The motivations and contributions are illustrated on a smart building example.

1 Introduction

Cyber-physical systems (CPSs) are widely used in many application domains. Guaranteeing their correct and proper behaviour is an essential and a challenging issue [11]. CPSs are smart systems that include engineered interacting networks of physical and computational components [30]. More and more devices are now organised as networked communicating systems with additional constraints on the whole system architecture, e.g., some robots put together and playing sounds more or less loudly depending on their respective positions. In this paper, we consider *structured* CPSs, composed of similar elements organized in motifs, responsible for carrying out some common functionalities. For example, a smart building composed of (modular) rooms and hallways is an example of CPSs with flexible and hierarchical structure, where a temperature regulation must be assured as a common functionality.

Managing (self-)adaptation is a topic of increasing attention and importance in software engineering [38], as it is crucial for functional as well as non-functional requirements, e.g., performance, power consumption or reliability. Component-based models allow dealing with both types of requirements while assuring correct-by-construction system development [15]. In addition, component-based models are well-suited to perform dynamic reconfigurations modifying system's architecture, as e.g. in BIP [5] and

⋆ This work was partially supported by the ANR grant ANR-23-CE25-0004 (ADAPT).
⋆⋆ O. Kouchnarenko was supported by the EIPHI Graduate School (grant number ANR-17-EURE-0002). This work was partially carried out during her research leave at Inria Lille.

DR-BIP [4]. However, a recent survey [13] emphasizes the need of a suitable methodology to ensure the correctness of reconfigurations in component-based systems (CBSs).

Self-adaptation of software elements has been extensively studied, notably using the formalism of the well-known MAPE loop [25], extended to MAPE-K with the knowledge management. Feedback control theory has recently emerged as a suitable adaptation methodology in line with the MAPE-K formulation [33]. The MAPE-K approach has however shown to favor monolithic controllers rather than structured controllers [14], that are more appropriate for (self-)adaptive software systems. Moreover, structured CPSs may have their implicit software regulators and explicit controllers interwoven, whereas in general, controller design is too strongly decoupled from the system under control. This is why designing and implementing more flexible controllers is challenging.

This paper aims to contribute to a flexible design and development of structured CPSs capable of (self-)adaptation to satisfy evolving internal and external constraints. This raises the following research questions:

RQ1 How to define a model accounting for the structural, behavioural and data aspects of CPSs that would be naturally amenable to hierarchical control?

RQ2 How to formulate the control problem of such structured CPSs with distributed elements?

To answer these questions, this paper's contribution consists in defining a notion of hierarchical motifs allowing the modelling of both the functional and the control components of CPSs.

The rest of the paper is organized as follows. Section 2 presents the control theory background, and Section 3 introduces the motivating example of a smart building temperature control. Section 4 introduces hierarchical motifs and illustrates this notion on the smart building example. Their composition semantics is presented in Sect. 5 (addressing **RQ1**). In Section 6, based on control motifs, the hierarchical control is expressed (tackling **RQ2**). Related work is presented in Section 7, and Section 8 concludes the paper.

2 On Control Theory

Control theory (CT) aims to stabilize and configure systems that evolve through time. CT is also a promising methodology for systems' (self-)adaptation [18, 23]. In particular, feedback controllers are algorithms that compute the adequate values for systems *knobs*, so that the *measures* meet their desired *reference* values. This decision making is repeated in a loop, allowing dealing with dynamic systems behaviour. In the control theory literature, knobs are also refereed to as control signals, actions or inputs, while measures can also been called outputs or sensors signals.

To control a system, the general methodology consists in (1) identifying a plant Ψ, e.g. the model of the system to be controlled, and (2) designing its associated controller C, the component that makes adaptation decisions to be applied to the plant [18]. Note that even if the systems considered are CPSs or software, *continuous* control is an adequate technique for self-adaptation [35]. Also, a *linear* control formulation has provided

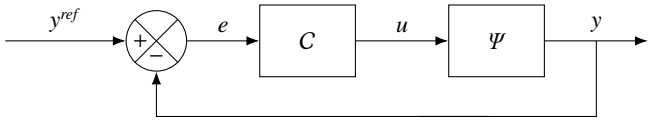


Fig. 1: Schema of a control feedback loop with notations

significant results even when used on complex and potentially non-linear software systems.

A block-schema representation of a feedback loop is given in Fig. 1: The plant Ψ captures the impact of changes of the values of knobs u on the measurements y, while the controller C sets the value of the knobs u based on the measurements y and their reference (e.g. objective value) y^{ref}, though the error e. Knobs and measures are signals that evolve with time, taking continuous (possibly quantified) values. Note that there can also be disturbances, modelled by external and uncontrolled signals that impact the system's behaviour.

Transfer functions Both the plant Ψ and the controller C are represented using transfer functions. A transfer function is a mathematical model representing the transformation of signals, e.g. how the knobs signal u affects the measurements signal y. One can thus write

$$y = \Psi u\,, \tag{1}$$

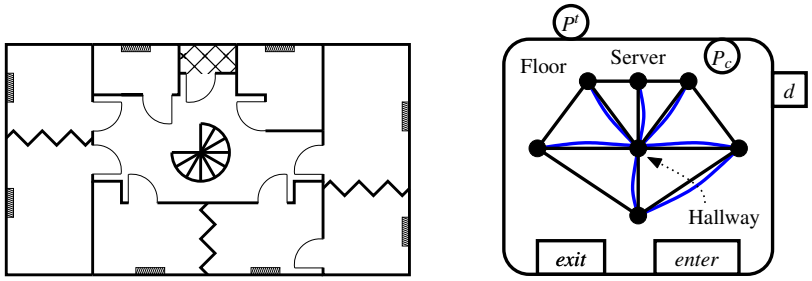
where a simple linear relation describes the system behaviour.

3 Motivating Example: Temperature Control in a Smart Building

To illustrate controlled CPSs with flexible and hierarchical structure, let us consider the temperature regulation of a smart building composed of rooms and hallways (see Fig. 2a). Some rooms are modular, meaning they can merge thanks to folding walls to form larger spaces. The rooms are interconnected due to temperature exchanges through walls, ceilings, and floors. Rooms are equipped with controllable heating systems. The target temperature for a room can vary depending on its type (e.g. office, corridor, server room) and occupation status (vacant or with people inside). Time or weather constraints (e.g. open/closed building, winter/summer) can also modify the temperature objectives.

The heat control is hierarchically structured, with (1) the lower level regulating the temperature in a single room, with one radiator each, (2) the middle level consisting of sets of modular rooms that can merge or split, (3) and the higher level setting temperature targets for the different rooms.

As the configuration of a room evolves, it changes room structure inside its set. The temperature regulation of such a smart building illustrates the control of flexible, hierarchically structured systems. For the following of the example, we build on the existing control formulations for building temperature regulation, using optimal control techniques for multi-room apartments [21, 22] and distributed control [29] for a multi-zone system, for which we add flexibility and hierarchical considerations.



(a) Floor plan (b) Floor motif

Fig. 2: One floor of the building (In (*a*), zigzag lines indicate foldable partition walls, the server room is shaded. In (*b*), double edges are for visual clarity only, they represent predicates on the underlying graph.)

4 Hierarchical Motifs

To address **RQ1**, the key element of our approach is the notion of motifs inspired by Dynamic Reconfigurable BIP (DR-BIP) [4]. In DR-BIP, the main role of motifs is to specify the interconnections among components throughout the lifetime of a system by relying on *maps*. In [4], maps are defined as "abstract concepts that denote arbitrary collections of inter-connected nodes (positions)".

In this paper, we adapt and extend the notion of motifs to allow a flexible design, hierarchical composition and control of *structured* CPSs. Moreover, as this paper focuses on hierarchical control, we rely on maps of the motifs to specify how measurements and control commands are propagated through the system hierarchy.For the purposes of this paper, a *map* is a directed graph with predicates on nodes and edges.

Definition 1 (Map)**.** *A map is a tuple* $\mu = (N, E, \mathcal{P})$*, where* N *is a* set of nodes*,* $E \subseteq N \times N$ *is a* set of edges*, and* $\mathcal{P}$ *is a set of predicates on* N *and* E*.*

For the sake of conciseness, we will write $n \in \mu$ and $(n_1, n_2) \in \mu$, meaning, respectively, $n \in N$ and $(n_1, n_2) \in E$, for $\mu = (N, E, \mathcal{P})$. We include a set of system-specific predicates to specify the nature of some nodes (e.g. centre of a star) or particularities of their arrangement (e.g. one node is located to the north of another).

Example 1. *Figure 2b shows a motif modelling the smart building floor shown in Fig. 2a. In particular, the motif comprises the corresponding map. The central node in the map represents the hallway. Other nodes represent the rooms (a modular room is also represented by a single node). We use two predicates (edge colours) to specify the possibility of heat transfer (black) and the presence of doors (blue) between nodes. Additionally, we use two node predicates:* `Server` *to identify the node representing the server room, and* `Hallway` *to identify the node representing the hallway.*

To be assembled hierarchically, motifs expose *interfaces*, which can be *external* (towards higher levels of the hierarchy) or *internal* (towards the sub-systems located in the nodes of the motif's map). An external interface provides an abstract view of

the state space of the sub-tree of the hierarchical assembly rooted in the motif w.r.t. both coordination (i.e. the set of *possible* discrete actions) and control (i.e. the set of *possible* measured values and the set of possible knob positions). Dually, an internal interface specifies the sets of *acceptable* actions, measurements and knob positions for the sub-tree rooted in a given node of the motif.

Let us assume given a universe of actions *Act*. Furthermore, let *Act* be a bounded lattice with the usual lattice operations $\vee$ (join) and $\wedge$ (meet), and bounds $\bot$ (bottom) and $\top$ (top).

Definition 2 (Interface)**.** *An interface is a tuple* $I = (A, S_Y, S_U, S_D, Y, U, D)$*, where* S_Y*,* S_U*, and* S_U *are vector spaces,* $A \subseteq Act$ *(with* $\bot \in A$ *and* $\top \notin A$*),* $Y \subseteq S_Y$*,* $U \subseteq S_U$*, and* $U \subseteq S_U$ *are, resp., the set of* actions *A and the domains of the* measurements*,* knobs *and the* disturbances *exposed by the interface.*

The product *of two interfaces is defined component-wise:* $I^1 \times I^2 \stackrel{def}{=} (A^1 \cup A^2, S^1_Y \times S^2_Y, S^1_U \times S^2_U, S^1_D \times S^2_D, Y^1 \times Y^2, U^1 \times U^2, D^1 \times D^2)$.

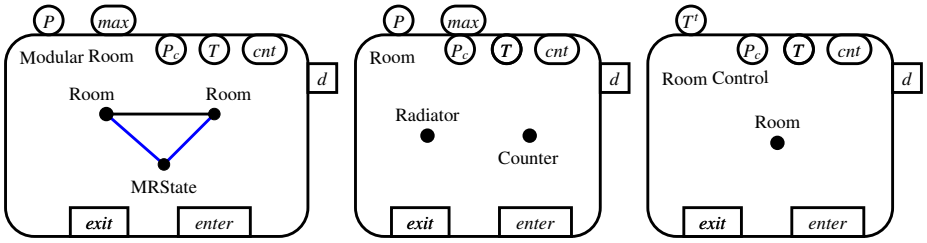
Clearly the product operation on interfaces is associative. Therefore, we will use the standard $\prod$-notation for products involving more than two interfaces. Although this operator is only commutative up to isomorphism due to the ordering of coordinates in the vector spaces, in this paper, we will only occasionally need an order of operands to be fixed. For the sake of clarity, we do not fix such an order explicitly but assume that *an* order is implicitly given and used consistently.

Example 2. *The external interface of the Floor motif is* $I_{Floor} = (\{enter, exit\}, \mathbb{R}, [0, MAX_{hs}], \mathbb{R}, [0, MAX_{hs}], \mathbb{R}^6, -)$*, with the actions enter and exit representing people arriving at and leaving the floor. The measure and knob* $P_c, P^t \in [0, MAX_{hs}] \subseteq \mathbb{R}$ *represent the power consumed since the previous measure and the target power consumed by the entire heating system of the floor (with* MAX_{hs}*—the maximal total power achievable—a parameter of the motif defined at instantiation). Finally, the disturbance* $d \in \mathbb{R}^6$ *(we omit the range for conciseness) represents the temperature of the room floor* T_f*, the walls* T_w*, the outdoor air* T_o *and the ground* T_g*, the solar radiation on the walls and windows* $\mathcal{R}_s$*.*

The internal interface of the Floor motif is the product of the external interfaces of its nodes, corresponding, resp., to the control of the 3 Modular Rooms, 3 Rooms, and one Hallway. The map of the Room Control motif (see Fig. 3c) only has one node corresponding to the controlled room.[3]

Example 3. *The external interfaces of the motives Modular Room (see Fig. 3a), Room (see Fig. 3b), and Hallway (not shown) coincide. That interface* $I_{Room} = (\{enter, exit\}, \mathbb{R} \times \mathbb{R} \times \mathbb{N}, [0, MAX_{hmr}] \times [-MAX_{tmr}, MAX_{tmr}] \times [0, MAX_{pmr}], \mathbb{R} \times \mathbb{N}, [0, MAX_{hmr}] \times [0, MAX_{pmr}], \mathbb{R}^6, -)$*, has two actions, enter and exit, as in the Floor motif. The measure* $P_c \in [0, MAX_{hmr}] \subseteq \mathbb{R}$ *is the same as for the Floor interface (with* $MAX_{hmr} \leq MAX_{hs}$*). The knob* $P \in [0, MAX_{hmr}] \subseteq \mathbb{R}$ *represents the power to be applied by the room radiator(s). The measure and the knob* $cnt, max \in [0, MAX_{pmr}] \subseteq \mathbb{N}$ *represent the current and the maximum admissible numbers of people. The measure* $T \in [-MAX_{tmr}, MAX_{tmr}] \subseteq \mathbb{R}$

[3] We use the same (control) motif for all the nodes of the Floor motif.



(a) Modular Room (b) Room (c) Room Control

Fig. 3: Modular Room, Room and Room Control motifs (Square boxes on the inside represent actions, on the outside—disturbances, rounded boxes on the inside represent measures, on the outside—knobs.)

represents the temperature of the room. The disturbance is the same as for the Floor motif.

The internal interface of the Modular Room motif is the product of the interfaces corresponding to the nodes of its map. For the Room nodes, these coincide with the external interface of the Room motif. For the singleton MRState node, the interface is $I_{MRState} = (\{merge, split, isMerged, isSplit\}, \{\cdot\}, \{\cdot\}, \{\cdot\}, \emptyset, \emptyset, \emptyset)$*, i.e. it has four actions and no measures, knobs, or disturbances. The actions correspond, resp., to merging/splitting the room and signalling its current state, i.e. whether it is merged or split.*

Finally, the external interface of the Room Control motif (see Fig. 3c) is the same as that of the Room motif, except for the knobs. Instead of $P \in [0, MAX_{hmr}] \subseteq \mathbb{R}$*, it has the knob* $T \in [-MAX_{tmr}, MAX_{tmr}] \subseteq \mathbb{R}$*, which sets the desired temperature within the controlled room.*

Observe Example 1 and Example 3 to notice that the role of the Floor motif is to derive target temperatures for each room based on the power consumption budget allocated to the floor. The controllers of each node are then responsible for achieving these temperatures by setting the power and the maximal capacity in their corresponding rooms.

Let us assume given two interfaces $I^{int} = (A^{int}, S_Y^{int}, S_U^{int}, S_D^{int}, Y^{int}, U^{int}, D^{int})$ and $I^{ext} = (A^{ext}, S_Y^{ext}, S_U^{ext}, S_D^{ext}, Y^{ext}, U^{ext}, D^{ext})$.

Definition 3 (Aggregation)**.** *An aggregation over* (I^{int}, I^{ext}) *is a mapping* $\mathcal{A} : 2^{A^{int}} \times S_Y^{int} \to S_Y^{ext}$*, such that* $\mathcal{A}(2^{A^{int}} \times Y^{int}) \subseteq Y^{ext}$*.*

An aggregation combines the measurements from the nodes of a motif and propagates the result to the next level of the hierarchy. Notice that the aggregation depends on the state of the motif represented by the set of enabled internal actions.

Example 4. *The external measure* P_c *of the Floor motif is the sum of the corresponding internal measures in the nodes:* $P_c = \sum_{n \in \mu} n.P_c$*.*[4] *This aggregation does not depend on the enabledness of the enter and exit actions.*

[4] The control interposed between the Floor and the Modular Room motifs propagates the measures from the plant up.

Example 5. *For the Modular Room motif, only the nodes corresponding to sub-rooms have to be taken into account for the power consumption aggregation (cf. Example 1):* $P_c = \sum_{n \in \mu \wedge \texttt{Room}(n)} n.P_c$.

Temperature measures provided by the sub-rooms can be aggregated to get the global temperature of a Modular Room. We define the aggregation as the simple averaging: $T = \frac{1}{\#\texttt{Room}} \sum_{n \in \mu \wedge \texttt{Room}(n)} n.T$, *where* #Room *denotes the number of nodes satisfying* Room. *More advanced processing based on room topology and the positions of the radiators can be used if necessary.*

Finally, the aggregation of the occupancy counters depends on the state of the modular room:[5] $cnt = \mathcal{A}(A_{en}^{MRState}, \dots)$, *with*

$$\mathcal{A}(A_{en}^{MRState}, \dots) = \begin{cases} \frac{1}{\#\texttt{Room}} \sum_{n \in \mu \wedge \texttt{Room}(n)} n.cnt & \text{if } isMerged \in A_{en}^{MRState}, \\ \sum_{n \in \mu \wedge \texttt{Room}(n)} n.cnt & \text{if } isSplit \in A_{en}^{MRState}. \end{cases}$$

Indeed, when a modular room is merged, it is impossible to know whether a person is located in one sub-room or another. Therefore, we make the counters in all sub-rooms mirror each other counting the total number of people in the modular room.

Definition 4 (Control profile)**.** *A control profile over* (I^{int}, I^{ext}) *is a mapping* $\Phi : 2^{A^{int}} \times S_Y^{int} \times S_U^{int} \times S_U^{ext} \times S_D^{ext} \to S_U^{int}$, *such that* $\Phi(2^{A^{int}} \times Y^{int} \times U^{int} \times U^{ext} \times D^{ext}) \subseteq U^{int}$.

A control profile specifies how the control command from the hierarchy (i.e. the external knob position) is propagated to the sub-nodes. Based on the control command and the current internal measurements and knob positions, it computes the updated knob positions at the nodes. Similarly to aggregations, profiles depend on the state of the corresponding motifs.

Example 6. *The control profile of the Floor motif sets the temperature target for each room depending on the total power consumption budget (external node) as follows:*

$$n.T^t = \begin{cases} 15, & \text{if } \texttt{Server}(n), \\ 18 + T_{cor}, & \text{if } \texttt{Hallway}(n), \\ 19 + T_{cor}, & \text{otherwise}, \end{cases} \quad \text{with} \quad T_{cor} = \begin{cases} 0, & \text{if } \sum_{n \in \mu} n.P_c < P^t, \\ 2 \cdot \dfrac{P^t - P_c}{P^t \cdot 10\%}, & \text{otherwise}. \end{cases}$$

If the total consumed power is below the budget, the temperatures are assigned according to the room profile. If power consumption exceeds budget, a correction is applied to reduce the target temperatures in the rooms and the hallway.

Example 7. *The control profile of the Modular Room depends on its state: when the room is merged, the same power is applied to the radiators in both sub-rooms, when it is split, the radiator power is set proportionally to the room capacity:*

$$\Phi_P(A_{en}^{MRState}, \dots) = \begin{cases} (P/2)^{n \in \mu \wedge \texttt{Room}(n)} & \text{if } isMerged \in A_{en}^{MRState}, \\ \left(\dots, \dfrac{n_i.MAX_{pmr}}{\sum_{n \in \mu \wedge \texttt{Room}(n)} n.MAX_{pmr}} \cdot P, \dots \right), & \text{if } isSplit \in A_{en}^{MRState}. \end{cases}$$

[5] See Example 10 in Section 5 for further detail.

Similarly, for the capacity control:

$$\Phi_{max}(A^{MRState}_{en}, \dots) = \begin{cases} (max)^{n\in\mu\wedge \texttt{Room}(n)} & \text{if } isMerged \in A^{MRState}_{en}, \\ \left(\dots, \dfrac{n_i.MAX_{pmr}}{\sum_{n\in\mu\wedge\texttt{Room}(n)} n.MAX_{pmr}} \cdot max, \dots\right) & \text{if } isSplit \in A^{MRState}_{en}. \end{cases}$$

(As in Example 5, the counters are mirrored when the room is merged.)

A disturbance profile specifies the disturbances at the nodes of the map based on the external disturbances and the internal measures.

Definition 5 (Disturbance profile)**.** *A disturbance profile over* (I^{int}, I^{ext}) *is a mapping* $\Delta : 2^{A^{int}} \times S^{int}_{Y} \times S^{ext}_{D} \to S^{int}_{D}$*, such that* $\Delta(2^{A^{int}} \times Y^{int} \times D^{ext}) \subseteq D^{int}$*.*

Example 8. *We omit precise description of the disturbance propagation for the sake of conciseness. Differentiating the temperature of the outdoor air on the four sides of each room, the disturbance profile of the Modular Room motif consists in assigning the temperature measure of each sub-room in the place of the outdoor air disturbance on the corresponding side of its adjacent sub-room(s). Similarly, the ambient temperature disturbance of the Radiator component is defined by the Room temperature measure.*[6]

Given a map $\mu = (N, E, \mathcal{P})$, and associated node and external interfaces $(I^n)_{n\in\mu}$ and I^{ext}, resp., coordination of the sub-system actions is specified using a First Order Logics (FOL).

Definition 6 (Interaction constraints)**.** *An* interaction logic $\mathcal{L}(\Sigma)$ *over* (μ, I^{int}, I^{ext})*, where* $I^{int} = (I^n)_{n\in\mu}$*, is an FOL with the signature* Σ*, such that*

$$\{\mathbf{E}\} \cup \{\mathbf{P}^p \,\big|\, p \in \mathcal{P}\} \cup \{\mathbf{a}^{ext}, \mathbf{a}^{int}, \mathbf{y}^{int}, \mathbf{u}^{ext}, \mathbf{d}^{ext}\} \subseteq \Sigma \tag{2}$$

with the usual FOL satisfaction relation $\models$*.*

The interpretation domain of $\mathcal{L}(\Sigma)$ *is as follows (we use* $\mathcal{I}$ *and* ν *to denote an interpretation of non-logical symbols and a valuation of variables, resp.):*

- *the* variables *range over the nodes of the map* μ*, i.e.* $\mathrm{codom}(\nu) = N$*,*
- *the predicate* $\mathcal{I}(\mathbf{E})(n_1, n_2) = ((n_1, n_2) \in E)$ *encodes graph connectivity in the map,*
- *symbols* $\mathbf{P}^p$ *are interpreted as the corresponding map predicates:* $\mathcal{I}(\mathbf{P}^p) = p$*,*
- *the constant* $\mathcal{I}(\mathbf{a}^{ext}) : \{\cdot\} \to A^{ext}$ *determines which external action is to be fired under the interpretation* $\mathcal{I}$*,*
- *the function* $\mathcal{I}(\mathbf{a}^{int}) : N \to \bigcup_{n\in N} A^n$*, such that* $\mathcal{I}(\mathbf{a}^{int})(n) \in A^n$*, for any* $n \in N$*, determines which internal actions are to be fired under the interpretation* $\mathcal{I}$*,*
- *the function* $\mathcal{I}(\mathbf{y}^{int}) : N \to \bigcup_{n\in N} Y^n$*, such that* $\mathcal{I}(\mathbf{y}^{int})(n) \in Y^n$*, for any* $n \in N$*, determines the internal measures,*
- *the constant* $\mathcal{I}(\mathbf{u}^{ext}) : \{\cdot\} \to U^{ext}$ *determines the external knob position,*
- *finally, the constant* $\mathcal{I}(\mathbf{d}^{ext}) : \{\cdot\} \to D^{ext}$ *determines the external disturbance value.*

[6] The Room temperature measure depends on the Radiator casing temperature measure, thereby creating a feedback loop affecting the radiator control.

An interaction constraint *over* (μ, I^{int}, I^{ext}) *is a formula* $\varphi \in \mathcal{L}(\Sigma)$*, such that* $(\bot, (\bot)_{n\in\mu}, y^{int}, u^{ext}, d^{ext}) \models \varphi$*, for any* $y^{int} \in Y^{int}$*,* $u^{ext} \in U^{ext}$*, and* $d^{ext} \in D^{ext}$*.*

Intuitively, the interaction constraint specifies what actions can be taken by each of the motif's sub-systems in view of the received measurements and control command and how these actions are combined to be exposed through the external interface.

Notice that we do not limit the signature of $\mathcal{L}(\Sigma)$ to the symbols explicitly stated in Definition 6. In particular, symbols can be included to represent internal knob positions or external measurements, or, alternatively, aggregation and profile mappings. Other symbols may represent additional non-persistent information, i.e. carrying values that are discarded from one interaction to another.

Example 9. *The interaction constraint defines the syncrhonisations of the internal and external actions of the Modular Room motif:*

$$\forall n(\texttt{Room}(n)),\ \Big((\mathbf{a}^{int}(n) = enter \implies \mathbf{a}^{ext} = enter) \wedge \tag{3}$$

$$\wedge\ (\mathbf{a}^{int}(n) = exit \implies \mathbf{a}^{ext} = exit)\Big) \wedge \tag{4}$$

$$\exists! n_{MRState} :\ \texttt{MRState}(n_{MRState}) \wedge \tag{5}$$

$$\mathbf{a}^{ext} \in \{enter, exit\} \implies \mathbf{a}^{int}(n_{MRState}) \in \{isMerged, isSplit\} \wedge \tag{6}$$

$$\mathbf{a}^{int}(n_{MRState}) = isMerged \implies$$

$$\Big(\mathbf{a}^{ext} = exit \implies \forall n(\texttt{Room}(n)), \mathbf{a}^{int}(n) = exit \tag{7}$$

$$\wedge\ \mathbf{a}^{ext} = enter \implies \forall n(\texttt{Room}(n)), \mathbf{a}^{int}(n) = enter\Big) \wedge$$

$$\mathbf{a}^{int}(n_{MRState}) = isSplit \implies$$

$$\Big(\mathbf{a}^{ext} = exit \implies \exists! n(\texttt{Room}(n)) : \mathbf{a}^{int}(n) = exit \tag{8}$$

$$\wedge\ \mathbf{a}^{ext} = enter \implies \exists! n(\texttt{Room}(n)) : \mathbf{a}^{int}(n) = enter\Big) \wedge$$

$$\Big(\mathbf{a}^{int}(n_{MRState}) \in \{merge, split\} \implies \forall n(\texttt{Room}(n)), \mathbf{y}^{int}(n).cnt = 0\Big). \tag{9}$$

Lines (3) and (4) above specify that a person can only enter or exit a sub-room if they, resp., enter or exit the modular room. Line (5) states that there is a node, denoted $n_{MRState}$*, that keeps track of the modular room's state. Line (6) states that when executing enter or exit of the room, the state signal must necessarily be consulted. If the room is merged (lines (7)), exiting or entering the modular room means doing so for* all *sub-rooms (cf. Examples 5 and 7). If the room is split (lines (8)), there must be exactly one sub-room on which the same action is performed. Finally, line (9) requires that the room be empty whenever its state is changed.*

Notice that the "signalling" actions isMerged and isSplit are not exported directly. They are only fired as part of interactions with enter and exit actions of the nodes. These interactions are exported as the corresponding external actions of the motif.

To summarise, a motif comprises all the elements introduced in Definitions 1 to 6.

Definition 7 (Motif)**.** *A motif is a tuple* $M \stackrel{def}{=} (\mu, (I^n)_{n\in\mu}, I^{ext}, \mathcal{A}, \Phi, \Delta, \varphi)$*, where* μ *is a map,* I^n *(for each* $n \in \mu$*) and* I^{ext} *are, resp.,* node *and* external *interfaces, such that*

$A^{ext} \subseteq Act$, $A^n \subseteq Act$, *for each* $n \in \mu$, *and every external action* $a \in A^{ext}$ *is a join of internal ones, i.e. there exists* $N \subseteq \mu$ *and* $a_n \in A^n$, *for each* $n \in N$, *such that* $a = \bigvee_{n \in N} a_n$.

Denote $I^{int} \stackrel{def}{=} \prod_{n \in \mu} I^n$ *the* internal *interface of the motif. The remaining four components,* $\mathcal{A}$, Φ, Δ, *and* φ, *are then, resp., an aggregation, a control profile, a disturbance profile, and an interaction constraint over* (μ, I^{int}, I^{ext}).

Internal measures Y^{int}, external knobs U^{ext}, and external disturbances D^{ext} can be construed as inputs of a motif. Dually, external measures Y^{ext}, internal knobs U^{int}, and internal disturbances D^{int} can be construed as its outputs.

5 Composition Semantics

In the context of structured CPSs, their model is a tree with motifs at all internal nodes and components defining the systems' behaviour at the leaves. Figure 4 shows a fragment of such a tree modelling the Smart Building example. The children of each internal node correspond to the nodes of the map of the motif. The flexibility of our approach lies with the fact that we do not restrict the nature of components, which may be instantiated motifs (see Definition 9 below), simple, timed or hybrid automata,[7] or any other kind of objects that have an operational semantics expressible as a Labelled Transition System (LTS) of the following kind.

Definition 8 (Object)**.** *An* object *implementing the interface* $(A, S_Y, S_U, S_D, Y, U, D)$ *is an entity that can be given an operational semantics in the form of an LTS defined by the transition relation* $\rightarrow \subseteq (2^A \times Y \times U) \times (A \times U \times D) \times (2^A \times Y \times U)$, *such that,*

- *for any transition* $((A^{en}, y, u), (a, \widetilde{u}, d), (A^{en\prime}, y', u')) \in \rightarrow$, *we have* $a \in A^{en}$ *and* $u' = \widetilde{u}$,
- *for any state* (A^{en}, y, u) *and any* $u' \in U$, $d \in D$, *there exist* $A^{en\prime}$, y' , *such that* $((A^{en}, y, u), (\bot, u', d), (A^{en\prime}, y', u')) \in \rightarrow$.

We write $(A^{en}, y, u) \xrightarrow{a, u', d} (A^{en\prime}, y', u')$ *to denote* $((A^{en}, y, u), (a, u', d), (A^{en\prime}, y', u')) \in \rightarrow$.

The state of an object is thus defined by the set of *enabled actions* and the current measurements and knob positions. The conditions imposed on the transition relation mean that (1) only enabled actions can be fired, (2) the knob positions can only be set externally and are not affected by the behaviour of the object, and (3) the bottom action is always enabled.

Example 10. *Radiator, Counter, and Modular Room State components shown at the leaves of the assembly tree in Fig. 4 are objects in the sense of Definition 8. The Modular Room State component type*[8] *implements the interface of the MRState node in the Modular Room motif (cf. Example 3). It has been extensively used in the examples of the previous section. It is worth noting the self-loop transitions labelled by the actions isMerged and isSplit. Their purpose is to signal that the component is, resp., in one of the states* `merged` *and* `split`.

[7] Our approach to modelling timed and hybrid aspects is based on [7, 8]. We do not present it here for the sake of conciseness.

[8] The full assembly comprises three instances—one for each Modular Room.

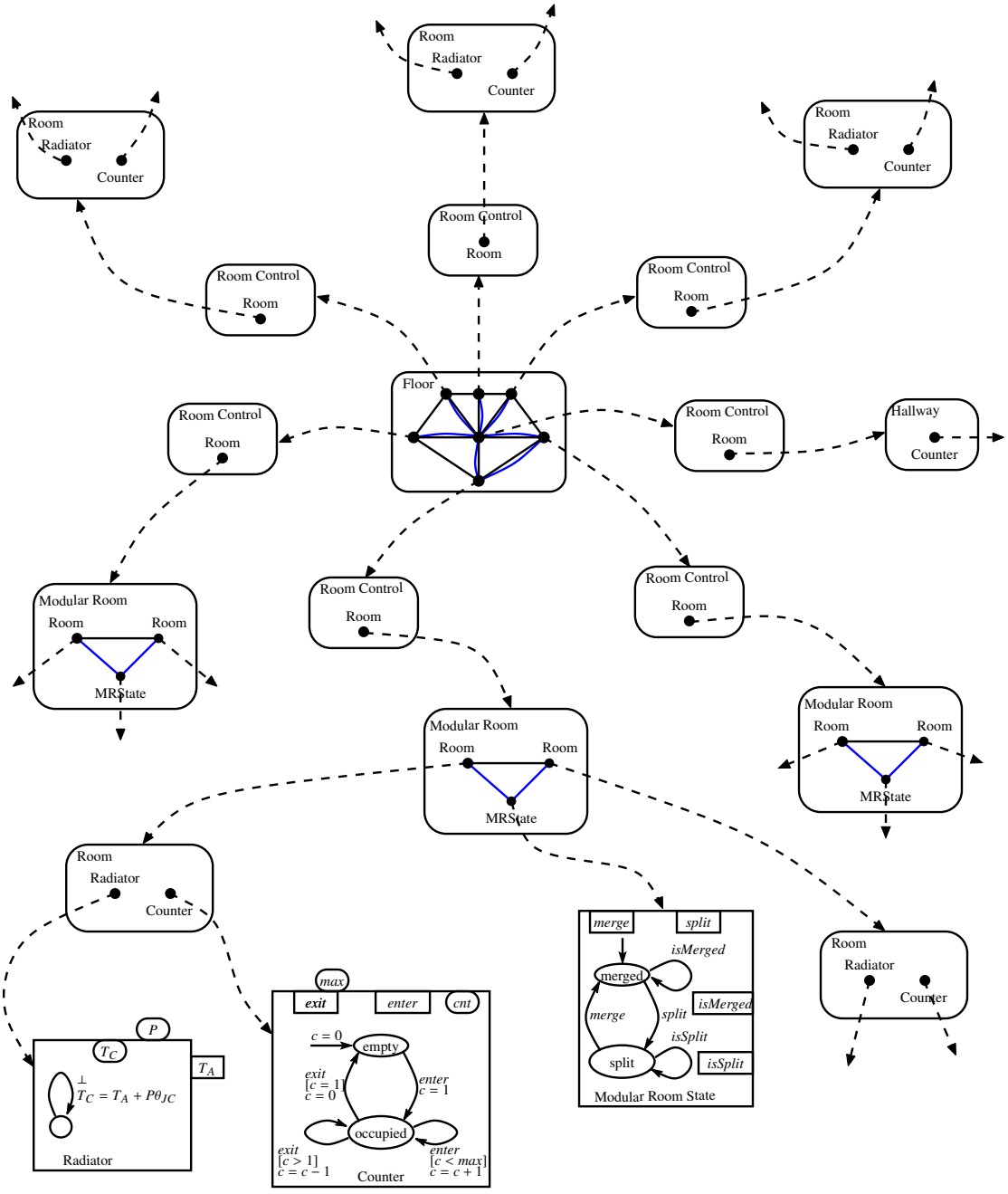


Fig. 4: Fragment of the Smart Building assembly (omitting explict modelling of time)

To allow hierarchical composition of motifs, we need to define what it means to instantiate a node of a motif with another one.

Definition 9 (Instantiation)**.** *A* motif instantiation *is a triple* $(M, N, \{O_n \mid n \in N\})$ *with* M *a motif,* $N \subseteq \mu_M$ *a set of nodes, and, for each node* $n \in N$*,* O_n *an object implementing the corresponding interface* I^n *of* M*. An instantiation is* closed *if* N *is the set of all nodes of* μ_M*. Otherwise, it is* open*.*

To allow the use of instantiated motifs as objects to, in turn, instantiate motifs at higher hierarchical levels, we must define their operational semantics. The semantics of a closed motif instantiation $(M, \mu_M, \{O_n \mid n \in \mu_M\})$ is given by the following Structural Operational Semantics (SOS) [32] rule.

$$\frac{\begin{array}{c} a \in A^{en} \qquad a = \bigvee_{n\in\mu} a_n \qquad u' \in U^{ext} \qquad (a, (a_n)_{n\in\mu}, (y_n)_{n\in\mu}, u) \models \varphi \\ (u'_n)_{n\in\mu} = \Phi\Big(\bigcup_{n\in\mu} A_n^{en}, (y_n)_{n\in\mu}, (u_n)_{n\in\mu}, u'\Big) \qquad (d_n)_{n\in\mu} = \Delta\Big(\bigcup_{n\in\mu} A_n^{en}, (y_n)_{n\in\mu}, d\Big) \\ \forall n \in \mu,\ (A_n^{en}, y_n, u_n) \xrightarrow{a_n, u'_n, d_n} (A_n^{en\prime}, y'_n, u'_n) \\ A^{en\prime} = \left\{ a' = \bigvee_{n\in\mu} a'_n \in A^{ext} \;\middle|\; a'_n \in A_n^{en\prime} \wedge (a', (a'_n)_{n\in\mu}, (y'_n)_{n\in\mu}, u') \models \varphi \right\} \\ y = \mathcal{A}\Big(\bigcup_{n\in\mu} A_n^{en}, (y_n)_{n\in\mu}\Big) \qquad y' = \mathcal{A}\Big(\bigcup_{n\in\mu} A_n^{en\prime}, (y'_n)_{n\in\mu}\Big) \end{array}}{(A^{en}, y, u) \xrightarrow{a, u', d} (A^{en\prime}, y', u')} \tag{10}$$

Informally, given an enabled joint action $a \in A^{en}$ (premises 1, 2) and a proposed position of the external knob u' (premise 3), such that a, the current internal measurements $(y_n)_{n\in\mu}$, and the current external knob position u satisfy the interaction constraint φ (premise 4), and the actuation of the internal knobs $(u'_n)_{n\in\mu}$ defined by the profile (premise 5) confronted with their disturbances $(d_n)_{n\in\mu}$ (premise 6) allow transitions within the objects at the nodes of the motif map (premises 7, 9), the motif instantiation can change its state accordingly (rule conclusion) with the set of enabled actions and the aggregation of the measurements in the target state being determined by the enabled actions, the proposed position of the external knob and the measurements exposed at the nodes of the motif (premises 8, 10).

Notice that, this semantics is an abstraction of the true behaviour of the object since it hides the internal measures. Therefore, subject to model checking, it can produce false positives. However, its execution in the context of simulation or control is driven in the bottom-up fashion using only the true values of the internal measures.

Proposition 1. *The semantics (10) defines an object in the sense of Definition 8.*

Proof sketch. We have to show that, in the composed LTS, (1) only enabled actions can be fired, (2) the knob positions can only be set externally and are not affected by the behaviour of the object, and (3) the bottom action is always enabled. The first two items follow trivially from premise 1 and the conclusion of the rule, resp. To show the third

item, consider $a = \bot$. Rule (10) reduces to

$$\frac{\begin{array}{c} u' \in U^{ext} \\ (u'_n)_{n\in\mu} = \Phi\Big(\bigcup_{n\in\mu} A_n^{en}, (y_n)_{n\in\mu}, (u_n)_{n\in\mu}, u'\Big) \qquad (d_n)_{n\in\mu} = \Delta\Big(\bigcup_{n\in\mu} A_n^{en}, (y_n)_{n\in\mu}, d\Big) \\ \forall n \in \mu,\ (A_n^{en}, y_n, u_n) \xrightarrow{\bot, u'_n, d_n} (A_n^{en\prime}, y'_n, u'_n) \\ A^{en\prime} = \left\{ a' = \bigvee_{n\in\mu} a'_n \in A^{ext} \;\middle|\; a'_n \in A_n^{en\prime} \wedge (a', (a'_n)_{n\in\mu}, (y'_n)_{n\in\mu}, u') \models \varphi \right\} \\ y = \mathcal{A}\Big(\bigcup_{n\in\mu} A_n^{en}, (y_n)_{n\in\mu}\Big) \qquad y' = \mathcal{A}\Big(\bigcup_{n\in\mu} A_n^{en}, y'_n\Big)_{n\in\mu} \end{array}}{(A^{en}, y, u) \xrightarrow{\bot, u', d'} (A^{en}, y', u')} \tag{11}$$

Indeed, from premise 2 of rule (10), we have $a_n = \bot$, for all $n \in \mu$. Since Definition 6 requires that the interaction constraint does not block $\bot$, the premise 4 is trivially satisfied. The premise 7 (premise 4 in rule (11)) is satisfied because the motif is instantiated with objects. All the remaining premises are non-blocking. □

This semantics is synchronous in the sense that transitions of both the motif and all the objects in its instantiation are taken atomically in one step. This is reflected by the last premises in (10) and (11), since the updated measurements y'_n are necessary to compute y'.

6 Control Protocols

This section focuses on linking the proposed modelling approach with the classical Control Theory. That link is necessary for defining control functions for the control motifs of the assembly. First we define control motifs—a special case of motifs defined in Section 4. Second, in order to address **RQ2**, we establish a link between the control formulation presented in Section 2 and the definition of motifs. Finally, we focus on hierarchical systems, establishing the plant formulation for instantiated motifs (i.e. subtrees of the assembly tree).

6.1 Control Motifs

Based on preliminaries from Section 2, we consider the basic control signals: measures y, reference values for these measures y^{ref}, and knobs u, a tunable signal that allows leveraging the measures signal. Some authors explicitly consider the error between reference and measure values. In our approach, it can be computed from these signals. In addition, we include the (external) disturbances in the control motif.

Consider an object implementing the interface $(A, S_Y, S_U, S_D, Y, U, D)$. Its control is realized by a *control function* $c : 2^A \times S_Y \times S_U \times S_Y \times S_D \to S_U$. The first parameter of the control function is the *control mode* determined by the set of enabled actions in the current state of the object. Given the control mode $A^{en} \subseteq A$, the current aggregated measurement $y \in Y$, the knob position $u \in U$, a reference value $y^{ref} \in Y$, and disturbance

$d \in D$, the control function defines the corresponding new value of the knob $u' = c(A^{en}, y, u, y^{ref}, d) \in U$ of the object.

We define *control motifs*, which are a special case of motifs in Definition 7 characterised by such control functions. We put $M_c = (\{\cdot\}, I^{int}, I^{ext}, id, c, id, true)$, where $\{\cdot\}$ is a singleton map (one node, no edges), $I_c^{int} = (A, S_Y, S_U, S_D, Y, U, D)$ (same as the interface implemented by the object), $I_c^{ext} = (A, S_Y, S_Y, S_D, Y, Y, D)$, $id : S_Y \to S_Y$ is the identity aggregation, the control function c plays the role of the control profile, and $id : S_D \to S_D$ is the identity disturbance profile. The use of the constant predicate *true* means that no interaction constraints are imposed by the motif. In the external interface of a control motif, the knob is replaced by the reference values of the measures. An control profile arising in the context of our running example is shown in Fig. 3c.

We do not impose any constraints on the nature of the control function. However, in the remainder of the paper, we consider linear controllers.

6.2 Linking Control Formulation & Motifs

The notions of plant, controller and their transfer function can be linked with the motifs as defined in Definition 7 and specified in Section 6.1.

Let us first consider a motif M in the most general case. The *plant* transfer function Ψ, as defined in Eq. (1), captures the impact of changes of knobs u on measurements y. It is a mathematical model linking elements in the motif's external interface I^{ext}, with $u \in U^{ext}$ and $y \in Y^{ext}$. The impact of disturbances d on measurements is also taken into account in the model, e.g. by artificially augmenting the measurement vector y with the disturbances.

Example 11. *The room plant Ψ_{room} is the model that represents the impact of the heating power on the temperature. It can be expressed as a multi-input multi-output model: Following Examples 2 and 3, the evolution of the Room motif temperature y_{room} (denoted T in Fig. 3b) can be computed based on the temperature of the floor T_f, of the walls T_w, of the outdoor air T_o and of the ground T_g, as well as the solar radiation on the walls and windows $\mathcal{R}_s$, and the radiation coming from the people in the room $cnt \cdot \mathcal{R}_o$, where $\mathcal{R}_o$ is the average radiation per person and cnt is the counter (see the Counter component in Fig. 4). The heat power control knob u_{room} (denoted P in Fig. 3b) is the internal heat flux in the room coming from the radiators.*

We define x as the set of all relevant disturbances (temperatures and radiations):

$$x \stackrel{def}{=} \left[y_{room}\ T_f\ T_w\ T_o\ T_g\ \mathcal{R}_s\ cnt \cdot \mathcal{R}_o \right]^T = \left[y_{room}\ d_{room} \right]^T . \tag{12}$$

Note that for a room connected to several others, T_w can be a vector. The evolution of indoor temperatures can be modelled as:

$$\begin{cases} x' = Ax + Bu_{room}, \\ y_{room} = Cx \end{cases} \tag{13}$$

where x' denotes either the derivative of x in the continuous-time case, or its value at the next time-step in the discrete-time case. [9] *A and B are matrices taking into account*

[9] Note that the conventional notation in control formulation is rather x^+.

the convection, thermal resistances, and capacity of the various elements around the room [10]*. The matrix C selects the room temperature among all the indoor ones:* $C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$.

The transfer function Ψ_{room} *is then classically computed based on the matrices of the model as:*

$$\Psi_{room}(s) = C(sI - A)^{-1}B, \tag{14}$$

with I the identity matrix of adequate size and s the complex variable. Note that the model takes into account the count of occupants in the room, to compute the induced radiation, and is thus a hybrid system, with both discrete- and continuous-state interfaces. However, in the following, we focus on linear continuous control, e.g. by considering all the elements of the state x as continuous in the control formulation, and let the hybrid control formulation as future work.

The plant Ψ is thus a partial view of a motif, only concerned with the external interface signals evolution. Its dependence with internal elements can be explored in the case where the map is specified, as presented in the next section.

We now consider a *control motif* M_c. The *controller* C captures the impact of the reference and the measures (often through their difference, i.e. the reference tracking error), and the disturbances d on the knobs signal :

$$u = C(y^{ref}, y, d). \tag{15}$$

Thus, the controller C models the link between the elements of the internal and external interfaces of the control motif M_c.

Example 12. *For our running example, the* ***controller*** C_{room} *computes the heat power knob value* u_{room} *based on the target temperature* y^{ref}_{room}*, the room measured temperature* y_{room} *and all the disturbances, that is the x vector. For the linear time-invariant system that we consider, the optimal controller can be computed as a state feedback, with precompensation for the reference tracking:*

$$u_{room} = -Kx + Gy^{ref}_{room}, \tag{16}$$

where K is the state feedback gain; and G is the precompensation gain, both being vectors of appropriate sizes. K is computed based on the plant Ψ_{room}*, more particularly on A and B, by pole placement. It allows specifying the desired closed-loop behavior, for instance the speed of reactivity of the control. The precompensation is computed based on A, B, C, and K, ensuring that the measure follows the reference. In this example, the controller* C_{room} *is thus composed of two transfer functions: K and G.*

Note that this controller is a state feedback (i.e. the control is computed based on x, meaning a measure of all its elements is needed), output feedback could rather be used (i.e. using only y_{room} *in the control formulation:* $u_{room} = -Ky_{room} + Gy^{ref}_{room}$ *) if the plant extended with the disturbances* d_{room} *is observable. In this case, a Luenberger observer or a Kalman filter could be used to estimate the full state x.*

[10] Computing the exact values of the A and B matrices is out of the scope for this work, the interested reader can refer to [21] for an example of those matrices.

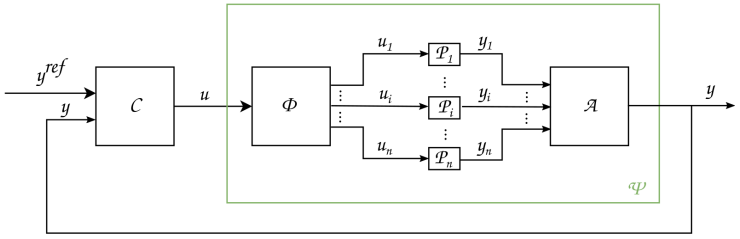


Fig. 5: Hierarchical control schema.

6.3 Hierarchical Control

We consider CPSs with a hierarchical structure, in which there is at least one controller. At a given level, the motif map and interfaces are instantiated, and we fix an order on the nodes of the map. A schematic representation of the control schema is given in Figure 5. Note that here the disturbances are not explicitly written in the following formulations for the sake of simplicity, as it can be considered as part of the measurement vector y (model extension with disturbance model).

The plant Ψ modeling the knobs-to-measures behaviour of the motif M, can be expressed using the control profile Φ, the aggregation $\mathcal{A}$, and recursion over the lower level motifs. At a given level, the measure $y \in Y^{ext}$ (external interface) is the aggregation of the measures of the lower levels y_i:

$$y = \mathcal{A}\mathbf{y}, \tag{17}$$

with $\mathbf{y} = [y_1, \cdots, y_i, \cdots, y_n]^T \in Y^{int}$ (internal interfaces). The controller C computes the knob $u \in U^{ext}$ (external interface), that is distributed among the lower levels as u_i by the profile function Φ:

$$\mathbf{u} = \Phi u, \tag{18}$$

with $\mathbf{u} = [u_1, \cdots, u_i, \cdots, u_n]^T \in U^{int}$ (internal interfaces).

The measure at a lower level can be derived from the value of the knob that was enforced, and is modelled by the subsystem transfer function P_i:

$$y_i = P_i u_i. \tag{19}$$

Note that if the lower level is not controlled, then $P_i = \Psi_i$. Otherwise, it is the transfer function of the controlled closed loop.

The hierarchical control consists then in designing C to regulate the plant Ψ, recursively formulated as:

$$\Psi = \mathcal{A}\mathbf{P}\Phi, \tag{20}$$

with $\mathbf{P} \stackrel{def}{=} [P_1, \cdots, P_i, \cdots, P_n] \times \mathbf{I}_n$.

Example 13. *Following Examples 11 and 12, let us now consider the hierarchical level of a modular room, composed of n rooms. The hierarchical control formulation allows deriving the model of the modular room Ψ_{mr} based on the room models $\Psi_{room,i}$, and on*

the aggregation and profile. At the lower levels, each room is controlled by a feedback controller, their equivalent closed-loop transfer function is thus:[11]

$$P_i = \frac{C_{room,i}\Psi_{room,i}}{1 + C_{room,i}\Psi_{room,i}}\,.$$

The modular room relevant temperature is the average temperatures in all rooms when it is merged, and the individual temperature measures otherwise. The aggregation $\mathcal{A}_{mr}$ can be written as:

$$\mathcal{A}_{mr} = \begin{cases} \begin{bmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix} & \textit{if state = merged,} \\ \boldsymbol{I}_n & \textit{if state = split.} \end{cases} \tag{21}$$

The profile Φ_{mr} distributes equally the temperature references, if we consider that all rooms are of similar use, and can be expressed as:

$$\Phi_{mr} = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T . \tag{22}$$

Overall, the modular room plant is then:

$$\Psi_{mr} = \mathcal{A}_{mr}\mathbf{P}_{room}\Phi_{mr}, \tag{23}$$

with $\mathbf{P}_{room} = [P_{room,1} \ \cdots \ P_{room,n}] \times \boldsymbol{I}_n$.

7 Related Work

On component-based models with layered architectures In this paper, only generic concepts of component-based systems are considered to allow applying the paper's proposals on hierarchical motifs to various component-based models, see e.g. a recent survey [13] for a list of component-based models. There are many approaches to model CBSs in general, and those supporting hierarchical style in particular [6, 10, 12, 16]. In [37], C2SADEL, a software architecture description and evolution language, defines C2-style architectures that can be seen as a network of concurrent components linked together by connectors, which are message routing devices. On its side π-ADL [31], which is formally derived from π-calculus, also allows defining architectural styles using it.

BIP [5] and DR-BIP [4] are suitable frameworks for developing CBSs with a layered structure. Some of these frameworks support monitoring and run-time property verification. However the hierarchical motifs integrating control that may usefully impact systems' architecture design and development and bring new verification and validation results, are original contributions of the present paper. Our use of motifs was inspired by DR-BIP. The results of this paper serve as a proof of concept aiming to implement a (DR-)BIP extension integrating hierarchical control motifs. Similarly, we use automata-based models of components and their underlying parallel and hierarchical compositions to represent the behavioural aspects of motifs and composed systems.

[11] Note that, for simplicity, this formulation assumes that the controller is based on output feedback, i.e. not state feedback and without precompensation.

It should be noted that, in a sense, our notion of instantiation is dual to the notion of *deployment* used in DR-BIP. Instead of specifying the nodes where each component is deployed, we specify objects, which may be components or motifs, that are located in nodes. Instantiation allows us to assemble motifs hierarchically. Hence, there is no need to have multiple components deployed on the same node. Instead, they can be arranged in an intermediate motif instantiating the node in question.

Fractal [10] allows defining a component assembly by linking their interfaces. Components' interfaces can be of two types dedicated either to the component content or to its membrane (control interfaces). Control interfaces allow building controllers, namely LifeCycleController, BindingController and AttributeController for primitive components, or ContentController for composite components. Standard control interfaces provided by Fractal components make it possible to create new components, modify the content of composites by adding or removing subcomponents, and to create or remove connections between interfaces. Differently from previous works, our approach provides the model of hierarchical motifs for both system's entities and their control. Dealing with attributes becomes possible at the composite level too thanks to motif's profile and aggregation functions. It greatly contributes to a flexible controllers design for systems with layered architectures.

Even more relevant is a less known work by Jean-Bernard Stefani [36], which introduced the G-Kells framework to describe dynamic structures with sharing. According to [36], G-Kells can be understood as an outgrowth of prior work by Stefani and Di Giusto [20], where they proposed a process calculus interpretation of the BIP model. As such G-Kells, DR-BIP and the hierarchical motif framework proposed in this paper share significant strands of genetic material. One common characteristic shared by G-Kells and our current proposal is the focus on the structuring elements of the system. Indeed, both approaches avoid specifying exactly the nature of the primitive behaviours, assuming only that their semantics can be defined in terms of a certain type of LTSs. Beyond that, one of the two principal characteristics of the G-Kells framework, arguably, is sharing: a given component (location) can be simultaneously attached to several other locations (e.g. a process can be a *functional component* of a larger system all while being *hosted* by a virtual machine). Moreover, no constraints are imposed on the attachment graph, which need not even be a Directed Acyclic Graph (DAG). In contrast, our current approach relies on a tree structure. In the future, we plan to generalise the approach to DAGs, which will require addressing the issue of conflict resolution, notably w.r.t. the control theoretic aspects of our work. The other principal characteristic of G-Kells is the structure dynamicity. While already addressed in DR-BIP, we leave the dynamicity in hierarchical motifs for future work.

On feedback control for software systems An overview on discrete-time control approaches for self-adaptive systems can be found in [15]. Control theory is a promising methodology for computing systems' (self-)adaptation [18,23]. In [33], a feedback control for both continuous-time and discrete-time cases has been related to the well-known MAPE-K loop in the framework of autonomic computing [25]. Control is then defined as a problem of restricting the uncontrolled system's behaviour, in order to enforce the desired behaviour, and avoid the undesirable one.

In [39], it is considered that MAPE addresses adaptation of software rather than physical properties or resources, whereas control theory (CT) loops are powerful at keeping some variables either at prescribed set points or within ranges, in the face of disturbances. [2] focuses on CPSs, trying to avoid that the issues in the software part affect the physical part. A safety controller is generated that a decision module can substitute to the complex controller to avoid violations of formal safety properties, in a sandboxing approach. Such a system augmentation is automatically generated thanks to reachability methods for hybrid systems. Hardware failure is not however considered. Controller synthesis for multi-agent control has recently been considered in [27], where signal temporal logic is used to express temporal behaviors of system, deriving continuous-time assume-guarantee contracts. Hierarchical organisation of systems is however not considered in this work.

In the model predictive control (MPC), the upper layer commands are fed to the lower levels adapting its behaviour when the conditions require such an action. Our work contributes to a conjecture in [39] by illustrating that in adaptive software the CT and MPC control scheme can be re-used, where the upper layer may be realized using MAPE. Finally, with relation to [26] focusing on brownout as opposed to blackout, the novelty of our approach consists in considering distributed or hierarchical control, and in handling different functionalities, i.e., in enabling a multi-variable control.

Formal methods for validating control systems Using formal methods for designing and validating systems' controllers with the aim to guarantee their desired properties, e.g. safety properties, is not new. However, as emphasized in [17], it is hard to formally verify properties of such feedback control systems.

In this context, used formal models are often focused on discrete time control part while abstracting continuous-time dynamics, at least partially. For example, static analysis techniques used in [3] for analysing automata modeling the control structure of synchronous embedded controllers, do not address continuous-time system control. In [1], theorem provers usefully provide static sufficient conditions for ensuring desirable safety properties for the closed-loop control designs. Differently from these approaches, we integrate the dynamics of the plants together with disturbances into the motif notion, in order to control and adapt hierarchical systems.

Verification of data-driven systems' representations controlled with feedback techniques is described in [17], where neural networks (NN) are used for their modeling with the aim to enforce properties such as reachability, safety, and stability of the feedback laws. This data-driven approach is promising, however in [17], systems with a layered control structure are not dealt with.

In [28] the authors aim to establish a common language to unify the study of architecture at different spatio-temporal scales. The proposed language for layered control architectures (LCAs) allows for a form of a hierarchy of control loops. Feedback control is however only considered at the lowest level, while other layers use other decision making techniques. Unlike this work, our approach allows modeling of structured systems with controllers potentially available at each level. For LCA systems, [24] introduces a new multiclock logic (MCL) to express assume-guarantee contracts, in order to prove global stability properties of a system using the stability properties of its components. Differently from [24], we use automata-based models of components and their

compositions. Our logic is used to express FOL interaction constraints among hierarchical motifs including control motifs, whereas MCL uses variables and clocks for assume-guarantee contracts at system-level and component-level verification.

In [34], the authors aim to verify properties of a broad class of continuous-time systems composed of interconnected components. The approach defines weak and strong semantics of assume-guarantee contracts for a compositional reasoning, where the week semantics is sufficient to deal with acyclic interconnections, and the strong one is required to reason on cyclic interconnections. In our framework, we aim to extend the class of the systems beyond those described by differential inclusions and invariance assume-guarantee contracts, where this strong-week semantics relationship applies.

8 Conclusion

This paper provided theoretical underpinnings to modeling both the system and its control by using hierarchical motifs with the aim to allow structured CPSs to be adaptive. More precisely, hierarchical motifs have been introduced in a control-compatible manner. We expect our paper to pave the way for the larger challenge of preserving control properties (convergence speed, transient response etc.), in addition to stability of feedback laws studied, e.g. in [17, 24].

As a future work direction, we intend to consider systems with discrete state changes, so that theory of control for hybrid systems would be necessary to study the behaviour of such systems, and the preservation of properties such as stability.

Behavioural refinement is another future work direction. We intend to exploit the notions of approximate simulation relations introduced in [19] that are suitable for safety critical systems' control and its refinement. In addition, we intend to integrate the interaction logic for parameterized systems in [9], which is decidable as it can be embedded in WSkS.

Finally, the key limitation of our approach is the tree structure of the considered hierarchical models. In future work, we intend to generalise to directed acyclic graphs by adding a conflict resolution layer in front of the motif's profile. Such a generalisation would allow the modelling of dynamically reconfigurable systems, where an object has to be redeployed from one node to another in a non-atomic manner.

References

1. N. Aréchiga, S. M. Loos, A. Platzer, and B. H. Krogh. Using theorem provers to guarantee closed-loop system properties. In *2012 American Control Conference (ACC)*, pages 3573–3580, 2012. ISSN: 2378-5861.
2. S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo. Sandboxing controllers for cyber-physical systems. In *IEEE/ACM Int. Conf. on Cyber-Physical Systems*, pages 3–12, 2011.
3. G. Balakrishnan, S. Sankaranarayanan, F. Ivančić, and A. Gupta. Refining the control structure of loops using static analysis. In *Proc. of the 7th ACM Int. Conf. on Embedded software*, EMSOFT '09, pages 49–58. ACM, 2009.
4. R. El Ballouli, S. Bensalem, M. Bozga, and J. Sifakis. Programming dynamic reconfigurable systems. In K. Bae and P. Csaba Ölveczky, editors, *Proc. Int. Conf. FACS 2018*, volume 11222 of *LNCS*, pages 118–136. Springer, 2018.

5. A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis. Rigorous component-based system design using the BIP framework. *IEEE Softw.*, 28(3):41–48, 2011.
6. F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Pérez. GCM: a grid extension to fractal for autonomous distributed components. *Ann. des Télécommunications*, 64(1-2):5–24, 2009.
7. S. Bliudze and S. Furic. An operational semantics for hybrid systems involving behavioral abstraction. In *Proceedings of the 10th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 693–706, Linköping, 2014. Linköping University Electronic Press, Linköpings universitet.
8. S. Bliudze, S. Furic, J. Sifakis, and A. Viel. Rigorous design of cyber-physical systems: Linking physicality and computation. *Int. J. on Software and System Modeling*, 18(3):1613–1636, 2019.
9. M. Bozga, J. Esparza, R. Iosif, J. Sifakis, and C. Welzel. Structural invariants for the verification of systems with parameterized architectures. In A. Biere and D. Parker, editors, *Proc. Int. Conf. TACAS 2020, Part I*, volume 12078 of *LNCS*, pages 228–246. Springer, 2020.
10. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal component model and its support in Java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11-12):1257–1284, September 2006.
11. T. Bures, R. Calinescu, and D. Weyns. Special issue on software engineering for trustworthy cyber-physical systems. *J. Syst. Softw.*, 178:110972, 2021.
12. T. Bures, P. Hnetynka, and F. Plásil. SOFA 2.0: Balancing advanced features in a hierarchical component model. In *Proc. Int. Conf. SERA 2006*, pages 40–48. IEEE Computer Society, 2006.
13. H. Coullon, L. Henrio, F. Loulergue, and S. Robillard. Component-based distributed software reconfiguration: A verification-oriented survey. *ACM Comput. Surv.*, 56(1):2:1–2:37, 2024.
14. R. de Lemos. Software self-adaptation and industry: Blame MAPE-K. In *Proc. Int. Symp. SEAMS 2023*, pages 88–89. IEEE, 2023.
15. R. de Lemos et al. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In *Software Engineering for Self-Adaptive Systems III. Assurances*, volume 9640 of *LNCS*, pages 3–30. Springer, 2013.
16. Z. Ding, Z. Chen, and J. Liu. A rigorous model of service component architecture. In G. Pu and V. Stolz, editors, *Proc. Int. Workshop TTSS 2007*, volume 207 of *ENTCS*, pages 33–48. Elsevier, 2007.
17. S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151–156, 2018.
18. A. Filieri et al. Software engineering meets control theory. In *Proc. IEEE/ACM Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, pages 71–82. IEEE, 2015.
19. A. Girard and G. J. Pappas. Hierarchical control using approximate simulation relations. In *Proc. of the 45th IEEE Conference on Decision and Control*, pages 264–269, 2006.
20. Cinzia Di Giusto and Jean-Bernard Stefani. Revisiting glue expressiveness in component-based systems. In *Proc. of the 13th International Conference on Coordination Models and Languages (COORDINATION 2011)*, volume 6721 of *LNCS*, pages 16–30. Springer, 2011.
21. I. Hazyuk, C. Ghiaus, and D. Penhouet. Optimal temperature control of intermittently heated buildings using model predictive control: Part I – Building modeling. *Building and Environment*, 51:379–387, 2012.
22. I. Hazyuk, C. Ghiaus, and D. Penhouet. Optimal temperature control of intermittently heated buildings using model predictive control: Part II – Control algorithm. *Building and Environment*, 51:388–394, 2012.

23. J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback control of computing systems*. John Wiley & Sons, 2004.
24. I. Incer, N. Csomay-Shanklin, A. D. Ames, and R. M. Murray. Layered control systems operating on multiple clocks. *IEEE Control Systems Letters*, 8:1211–1216, 2024.
25. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
26. C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez. Brownout: Building more robust cloud applications. In P. Jalote, L. C. Briand, and A. van der Hoek, editors, *Proc. Int. Conf. ICSE'14*, pages 700–711. ACM, 2014.
27. S. Liu, A. Saoud, and D. V. Dimarogonas. Controller synthesis of collaborative signal temporal logic tasks for multi-agent systems via assume-guarantee contracts. *IEEE Transactions on Automatic Control*, pages 1–16, 2025.
28. N. Matni, A. D. Ames, and J. C. Doyle. A quantitative framework for layered multirate control: Toward a theory of control architecture. *IEEE Control Systems Magazine*, 44(3):52–94, 2024.
29. P.-D. Moroşan, R. Bourdais, D. Dumur, and J. Buisson. Building temperature regulation using a distributed model predictive control. *Energy and Buildings*, 42(9):1445–1452, 2010.
30. National Institute of Standards and Technology (NIST, USA). Framework for cyber-physical systems (special publication 1500-201), 2017.
31. F. Oquendo. pi-ADL: an architecture description language based on the higher-order typed pi-calculus for specifying dynamic and mobile software architectures. *ACM SIGSOFT Softw. Eng. Notes*, 29(3):1–14, 2004.
32. G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
33. É. Rutten, N. Marchand, and D. Simon. Feedback control as MAPE-K loop in autonomic computing. In R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, editors, *Software Engineering for Self-Adaptive Systems III. Assurances*, volume 9640 of *LNCS*, pages 349–373. Springer, 2013.
34. A. Saoud, A. Girard, and L. Fribourg. Assume-guarantee contracts for continuous-time systems. *Automatica*, 134:1–13, 2021. Article 109910.
35. S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio. Control-theoretical software adaptation: A systematic literature review. *IEEE Transactions on Software Engineering*, 44(8):784–810, 2018.
36. Jean-Bernard Stefani. Components as location graphs. In *Revised Selected Papers of the 11th International Symposium on Formal Aspects of Component Software (FACS 2014)*, volume 8997 of *LNCS*, pages 3–23. Springer, 2014.
37. R.N. Taylor, N. Medvidovic, K.M. Anderson, E.J. Whitehead, J.E. Robbins, K.A. Nies, P. Oreizy, and D.L. Dubrow. A component- and message-based architectural style for GUI software. *IEEE Transactions on Software Engineering*, 22(6):390–406, 1996.
38. D. Weyns. Software engineering of self-adaptive systems. In S. Cha, R. N. Taylor, and K. Chul Kang, editors, *Handbook of Software Engineering*, pages 399–443. Springer, 2019.
39. D. Weyns et al. Towards better adaptive systems by combining MAPE, control theory, and machine learning. In *Proc. Int. Symp. SEAMS@ICSE 2021*, pages 217–223. IEEE, 2021.