Self-integrating Organic Control Systems: from Crayfish to Smart Homes

Ada Diaconescu
Telecom ParisTech, LTCI, IMT, FR
ada.diaconescu@telecom-paristech.fr

Pembe Mata
Telecom ParisTech, IMT, FR
matapembe@gmail.com

Kirstie Bellman
Topcy House Consulting, USA
bellmanhome@yahoo.com

***Abstract*—Survival in complex environments, for both natural and artificial systems, requires behavioural *adaptation* to common changes and behavioural *innovation* to face the unexpected. The challenge here is to produce a vast variety of behaviours, each adapted to current circumstances, while relying on a limited amount of resources (e.g. sensors, controllers and actuators), within a 'suitable' time-frame. Drawing inspiration from neural and behavioural studies on crayfish, this position paper brings to the fore several key design features that enable organisms to address this challenge. It then proposes a similar design for artificial controllers, based on: i) an extensible set of reusable control units; and, ii) a goal-driven, context-sensitive (self-)integration process for assembling control units into a wide variety of integrated system controllers. Pre-integrated sub-controllers can also be merged, to improve efficiency while avoiding conflicts. The proposal is illustrated via a proof-of-concept implementation for the smart home, where users can add and remove goals and devices at runtime and the controller is adapted accordingly. This study brings us closer to our long-term objective of defining reusable methodologies and platforms for the development of self-* systems running in complex unpredictable environments, notably including smart homes, cities, vehicular networks and electrical grids, merged via the Internet of Things, and of People.**

***Index Terms*—adaptive self-integration, control systems, goals, self-optimisation, conflict resolution, bio-inspiration, smart home**

I. Introduction

Survival in complex competitive environments, for both natural and artificial systems, requires behavioural *adaptation* to common changes and behavioural *innovation* to face the unexpected. Yet, in complex environments, the number of possible situations is rather vast and hence predicting all of them is impossible. Moreover, the amount of internal resources (e.g. sensors, controllers and actuators) available to a system is necessarily limited, and hence, developing and maintaining *separate* control mechanisms for producing correspondingly vast varieties of behaviours is infeasible. The resulting challenge is two-fold: i) how to produce a vast variety of behaviours from limited resources?; and ii) how to produce an adapted behaviour in each situation within a 'limited' time?

This position paper aims to tackle this challenge. Firstly, it relies on neural and behavioural studies on crayfish to highlight three key design features of nervous control systems. Secondly, it shows how similar designs can be adopted for developing artificial control systems, and uses smart homes as an illustrative application domain.

Part of this work has been developed through the joint research laboratory between Telecom ParisTech and EDF Labs – SEIDO-II

The essential design features observed in crayfish are: i) *neural configuration*: reusability of neurons with different output configurations for producing different behaviours when integrated with other neurons; ii) *behavioural integration*: reusability of pre-integrated neural circuitries that can merge spontaneously for producing composed behaviours; and, iii) *reafference principle*: enabling afferent signals (i.e. reference inputs) to be achieved by context-sensitive neural feedback loops. Notably, for efficiency reasons, neural circuitries are organised in a hierarchical manner, with higher-level neurons mediating and controlling the actions of lower-level ones. Conflicts are handled via inhibition of neural outputs.

Based on these principles, we propose a design for artificial controller based on: i) *control units (CUs)*: reusable control functions that can be configured to produce different control behaviours when integrated with other CUs; ii) *integrated controllers (ICs)*: reusable assemblies of CUs that can be merged to produce more complicated behaviours; iii) *goal-orientation*: (self-)integration of CUs into ICs, and of ICs into more complex ICs is driven by stakeholder goals (reference inputs). ICs are organised in a hierarchical manner, and conflicts managed via special-purpose resolution functions.

The key advantage of this design consists in the wide variety of control behaviours that it can produce from limited resource sets. Generating control behaviours that are well-adapted to their environment, and to their goals, is achieved: via *pre-wired integration* in common cases; and, via *trial-and-error learning-based integration* for unexpected cases. More advanced learning techniques, e.g. prediction-based, will be studied in future work. Once acquired by learning, new behaviours can be reused (as reflexes) in similar contexts.

We implemented a proof-of-concept smart home controller to illustrate these capabilities. The controller self-adapts to the runtime addition and removal of user goals (e.g. temperature, security and energy costs) and of 'smart' devices (e.g. heaters, thermometers, window blinds) by integrating pre-existing software modules (CUs) and behavioural composites (ICs), while handling conflicts. *Integration* here implies interconnection, configuration, communication and time-coordination among CUs, which may be added and removed during runtime.

The initial success of this prototype supports the claim that the proposed design enables the integration of pre-

wired control behaviours for handling known situations, with dynamically-learned composite behaviours for dealing with new circumstances. Even if the learning process in this initial implementation is rather basic – e.g. CUs added by hand during runtime and implemented to ensure successful integration with existing CUs – we show that the underlying design provides the needed organisation and runtime changes for context-dependent adaptations. These results pave the way for achieving reusable platforms for developping viable control systems for complex unpredictable environments.

II. Adaptive Neural Integration in Crayfish

A. Overview of key properties

We consider crayfish as a representative example of invertebrate organism with relatively simple nervous system, which, according to numerous studies (e.g. [1]–[3]), can produce a wide range of behaviours (e.g. types of movement) in response to various situations (e.g. combinations of external stimuli and internal state); and learn new behaviours in new situations. Interestingly, this wide behavioural diversity is based on a limited number of neurons, sensory and motor components. The decisive underlying characteristic is the ability to generate complex behaviours (e.g. escaping a threat) from an ordered sequence of simpler behavioural units, of general purpose (e.g. various abdomen contractions and tail flips for leaping and swimming). This can be achieved via a hierarchical neural organisation, where 'higher-level' neural circuitry (i.e. coordination) can adapt, in a context-sensitive manner, the *integration* of 'lower-level' neural circuitry (i.e. control of simpler movements), leading to various composite behaviours.

Three key design features can be identified here:

- *Neural configuration*: studied at the neural level [2], this design feature consists in reusing individual neurons with different *configurations*, for producing different outputs, in different contexts. 'Configuration' is achieved via inhibitions from 'higher-level' context-sensitive neurons, for preventing conflicts with other active elements, and via excitation for supporting the co-occurrence among compatible elements. This improves efficiency by reusing control resources to produce basic behavioural variants.
- *Behavioural integration*: studied at the behavioural level [1], [3], this design feature enables merging of individual neural circuitries for obtaining composite behaviours in complex situations. Taken separately, each neural circuitry produces a relatively simple behaviour in response to common stimuli. Merged behaviours can provide more complex responses and even react to new combinations of stimuli. This also improves efficiency by reusing pre-integrated behaviours to produce new ones.
- *Reafference principle*: studied at both neural and behavioural levels [4], this design feature enables input signals from 'higher-level' neurons to be enacted irrespectively of a changing context (e.g. a fish maintaining its position despite water currents that tend to displace it). Input signals are enacted via 'lower-level' neural circuitry that adapts its *efferent* signals (e.g. commands to muscles), based on *afferent* feedback (e.g. signals from muscles and environment) – similarly to feedback in 'classic' control theory.

We look into these mechanisms in more detail next.

B. Adaptive neural configuration

To illustrate this mechanism, we focus on the way in which, in crayfish, a pair of pre-motor inter-neurons, the I3's, are engaged in different types of tail-flips during various escape actions [2]. For the purpose of our discussion we will focus on how the I3's dendrites (i.e. "inputs") can be excited by three kinds of command neurons: i) the medial giant (MG) neurons – for escaping from frontal threats via a backward leap; ii) the lateral giant (LG) neurons – for escaping from threats around the tail via a somersault; and, iii) non-giant (NonG), 'voluntary', circuitry – for swimming away from less urgent threats. The MG and LG mediate rapid stereotyped reactions to sudden threats; while the NonG mediate aleatory trajectories in response to progressively developing threats.

These three types of escape movements – backward or forward leaps, and aleatory swimming – are achieved via different types abdomen contractions and complementary kinds of tail flips. These must be configured to produce coherent movement. The I3's axons (i.e. neural "outputs") connect to the posterior and ventral telson flexor (VTF) motoneurons, which are in turn connected to the crayfish VTF muscles, located in the tail fan. Contracting the VTF muscles produces tail flips that help with backward movement. These are suitable for escaping from forward threats, but not from backward threats. Hence, the I3's should be inhibited in the latter case. However, the I3's axons also connect, indirectly, to neurons that inhibit afferent nerves (AFF) located in the tail fan. This afferent inhibition cancels reafference, which would otherwise be caused by the sudden vigorous tail-flips and would hamper rapid escape. Hence, the I3's should be recruited.

We notice that during backward escape the I3's activation is subject to *conflict* – there is a need to only activate the I3's afferent suppression signals while *not* activating its contraction of the VTF muscles. The LG neurons 'resolve' this conflict by: 1) exciting the I3's; and, 2) inhibiting their connection to the VTF neurons (Figure 1-a). The timing between these two actions is such that the LG's inhibitory signal to the VTF neurons (1.5ms) is faster than the I3's excitatory signal to the VTF neurons (2.4ms). The MG inhibits the I3's in a similar manner. Finally, the NonG circuitry recruits the I3's for voluntary movements that involve the VTF muscles.

Based on these facts, the I3's neurons can be considered as a basic behaviour controller, which can be recruited by 'higher-level' controllers (i.e. LG, MG and NonG neurons) to generate different behavioural patterns, via different configurations.

C. Adaptive integration of neural circuitry

Composite behaviours are produced by integrating several neural circuitries, each one generating a simpler behaviour, or movement. We study two cases here: one for producing

composed escape behaviours, and one for combining escape and feeding behaviours.

In crayfish, escape from sudden backward threats involves an immediate vigorous tail-flip, causing a forward leap, followed by a sequence of rapid tail-flips, enabling swimming [3]. The immediate reaction is mediated (or triggered) by the LG neurons and swimming by NonG circuitry (Cf. subsection II-B). Timing is essential here for achieving the desired *sequencing* of these movements. Studies have found [3] that the two types of neurons – LG and NonG – are triggered at the same time, by the same (threat) stimuli, yet their reaction times differ by at least an order of magnitude. Namely, the muscles involved in the forward leap are activated first (i.e. about 6ms after the threat stimulus, via LG) and the ones involved in swimming (i.e. after about 50-500ms, via NonG).

Further behavioural composition can occur in crayfish from already composite behaviours, like escaping and feeding. These common behaviours are triggered by different types of stimuli, like food availability and threats, respectively. Often, these behaviours entail different usages of (the same or of different) body parts, which are incompatible if triggered simultaneously – e.g. different types of tail-flips, as discussed in II-B; or, grasping heavy loads while swimming. Hence, activating several such behaviours simultaneously, in response to parallel stimuli, would lead to *conflict*.

The ability to choose among several behaviours that are suitable within a given context is a major accomplishment of the nervous system, for avoiding conflict. Often, alternative behaviours inhibit one another, in a context-sensitive way. E.g., when crayfish feed on large pieces of food, either escape is inhibited, or the food is released (feeding is inhibited) [1].

These observations suggested the existence of neural pathways that cross-connect the circuitry that mediates different behaviours. Such coordination pathways could hence provide a relatively direct way for behaviour selection and inhibition. Surprisingly, while these assumptions apply in certain contexts, e.g., when one behaviour is significantly superior to the others, in more complicated contexts, where several behaviours may be simultaneously viable, more sophisticated behaviours were observed [1]. Namely, if crayfish perceived a threat during feeding on small pieces of food then usually it carried these away – hence merging feeding and escape behaviours. Moreover, the probability of escape was higher than when large food pieces were consumed. Hence, rather than selecting and inhibiting alternative behaviours, the alternative behaviours were recomposed from simpler motions, reacting to most stimuli, and avoiding conflict – e.g. escaping while holding onto food, but not consuming it.

These findings show that neural coordination of control mechanisms, even in relatively primitive invertebrates, is more sophisticated than the mere selection of pre-wired behaviours; and can synthesise complex behaviours via context-dependent adaptation and integration of simpler behaviours.

D. Control feedback via reafference

The reafference principle arose from the insight that many reflexes were not simple input-output processes, but involved continuous activity in neural circuitry [4]. This explained, for instance, how fish maintained not only 'normal' postures (e.g. horizontal), but also 'abnormal' postures (e.g. vertical or sideways), both via reflex actions correcting deviations from the 'normal' or 'abnormal' positions, respectively. This suggests that reflexes react not only to external inputs but also to internal inputs from higher-level neurons (i.e. afferent signals). In short, higher efferent signals for voluntary movement are transmitted as lower efferent signals to muscles; afferent signals from the muscles are compared to the higher efferent signals; and, the lower efferent signals to muscles are updated accordingly, to minimise the difference between higher afferent signals and the efferent signals (Figure 1-b).

The reafference mechanism is equivalent to negative feedback loops in 'classic' Control Theory: higher afferent signals are equivalent to Reference control inputs, efferent signals to monitored outputs from the controlled system, and lower afferent signals to adjusted inputs into the controlled system. It also resembles feedback loops in self-* systems, with higher afference signals representing the goals of the self-* systems.

III. Reusable Design for Goal-Oriented (Self-)Integration of Artificial Controllers

A. Design Overview

Drawing inspiration from the neural control design concepts discussed above (II) we propose a design approach for artificial controllers. This proposal fits within our previous work on generic goal-oriented holonic architectures, e.g. [5] and [6]. The contribution in this paper focuses on the specifics of the *controller self-integration process*; and on the way it handles *conflict resolution* between incompatible control behaviours. Importantly, self-integration reuses pre-integrated (sub-)controllers, when possible, for better efficiency.

In short, a controller receives input *goals* from stakeholders (e.g. users, or other controllers) and produces corresponding output *actions* on system resources for achieving the goals (Figs. 2 and 3). It then *monitors* the results of its actions and accordingly *adapts* its actions to better attain the goals. It also provides *goal evaluations* to the stakeholder. We note that actions can be viewed as lower-level goals, and monitoring as lower-level evaluation. This view is consistent with the generic architectures of Autonomic and Organic Computing,

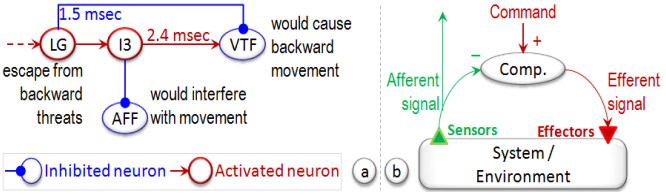

Fig. 1. a) Ex. of adaptive neural configuration via excitement & inhibition; b) Reafference principle in control theoretic terms, based on [4] and [18]

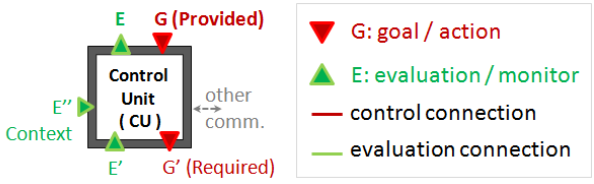

Fig. 2. Control Unit (CU) with Provided / Required Goals and Evaluations

with 'classic' control systems (i.e. goals as Reference inputs), and with reafference (II-D) (i.e. goals as afferent signals).

We further decompose this generic controller into a set of sub-controllers, recursively, until reaching elementary parts, called *Control Units* (CUs) – e.g. software modules, services or components – Fig. 2. CUs can be added, removed and (self-)integrated at runtime, hence producing a wide variety of *Integrated Controllers* (ICs). All CUs and ICs fit the generic goal-oriented self-adaptive design described above. Fig. 3 illustrates a basic IC, composed of two CUs: CU_A and CU_B. A CU may implement basic control functions, such as monitoring, decision-making or acting; entire feedback-loops, such as self-optimisation; conflict-resolution algorithms; learning, and so on – complete CU typology in future work.

Our aim here is two-fold. Firstly, we aim to enable CUs to *(self-)integrate*, in various ways, to produce a large number of IC variants, featuring composite control behaviours that can respond to a wide range of runtime changes – in the context, goals and resources. An optimisation problem here is to strive for the best IC composition that can meet the goals. Secondly, we aim to ensure the *scalability* of the self-integration process – i.e. efficiency and viability for large numbers of CUs and high frequencies of change. The problem is to find a 'satisficing' controller, which meets the goals without being necessarily optimal, within an 'acceptable' time.

To reach the first objective (self-integration) we extend a 'classic' *service-oriented component (SoC)* approach [12]. Here, a CU is a service-component that *provides* and *requires* goals (services) to/from other CUs. *Dynamic service binding* connects CUs by matching their required and provided goals (i.e. input and output types). However, unlike services, CUs are autonomous entities that may accept or reject requests for

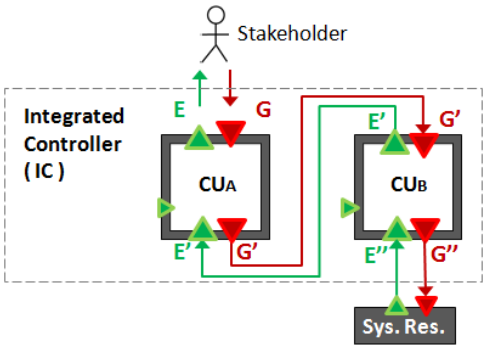

Fig. 3. Integrated Controller (IC) with two Control Units: CU_A and CU_B

providing their goals (e.g. to avoid conflicts with other goals or prevent resource contention). A simple way to achieve this is via a goal request/reply protocol over interconnected CUs. CUs that accept goal requests form an IC. Moreover, unlike SoC applications, ICs are *evaluated* with respect to their ability to reach their goals. Unsuccessful ICs are dismantled, re-triggering the self-integration process (details in III-B).

To reach the second objective (scalability) we organise the self-integration process in a *hierarchical* manner. This allows *pre-integrated* CUs ($sub-ICs$) to be reused, even if sub-optimal; rather than re-integrating new ones from scratch.

We note the following similarities between the proposed design and the observed design of studied nervous systems:

- Adaptive neural configuration, by output inhibiting (II-B): adaptive CU configuration, by action blocking (III-B);
- Adaptive integration of hierarchical neural circuitry (II-C): adaptive (self-)integration of hierarchical CUs and/or $sub-ICs$ into larger ICs (III-B);
- Neural reafference principle (II-D): goal-oriented controller (self-)integration and system control (III-B).

We look into the design details of these processes next.

B. Goal-oriented Controller Integration and System Control

Integration involves several complementary operations: find, interconnect, configure, execute and coordinate CUs; and, detect and resolve conflicts. Each CU, $sub-IC$ and IC is evaluated with respect to its ability to reach its goals. In case of negative evaluation CUs may be discarded (and replaced) and $(sub-)ICs$ may be dismantled and reintegrated dynamically – partially or fully, involving (some of) the operations above.

Controller integration proceeds in a top-down manner, starting from high-level stakeholder goals, progressively refining these into sub-goals, and finally into low-level actions on system resources. For each (sub-)goal or action, a CU or $(sub-)IC$ that provides that goal/action must be found and recruited. Once functional, each $(sub-)IC$ is evaluated (bottom-up) and only retained if successful. Hence, integration may follow several top-down and bottom-up iterations ('yoyo' process, e.g. [13]) before finding a suitable IC (Cf. [5]).

In Fig. 3 for instance, (self-)integration starts when a stakeholder requests goal G on CU_A, which provides G. If CU_A accepts the request to provide G, then it must find a provider for its requested goal G'. Hence, CU_A finds CU_B and requests it to provide G'. CU_B requires to perform actions G'' on resources $Sys.Res.$. When this connection is formed, the integration process stops.

Once the resulting IC is operational, it is evaluated with respect to the goals it accepted. Namely, CU_B evaluates how $Sys.Res.$ provision G'' – negative results prompt CU_B to select alternative providers for G''. If G'' is provided correctly, CU_B also evaluates its ability to achieve G', and may accordingly adjust its internal operation and/or its request for G''. Similarly, CU_A evaluates CU_B for G', and may accordingly choose another G' provider, CU'_B, and/or adapt G'. Finally, the stakeholder evaluates CU_A for G, and may choose another G provider, CU'_A, or update G.

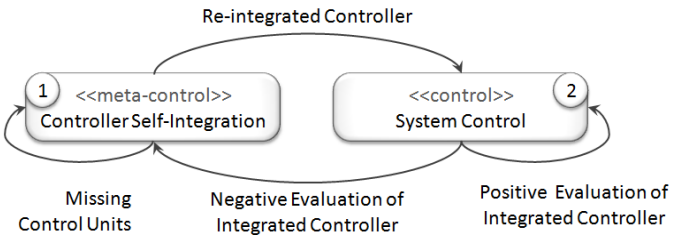

Fig. 4. State Diagram of a Self-integrating Controller

Selecting an alternative goal provider, at any level, may trigger another top-down integration process, with this provider at the top. E.g., if CU'_B requires G''' then it must find a provider for it, and so on. This top-down and bottom-up integration process ('yoyo') should eventually stabilise (or oscillate over relatively 'large' periods) – future work. For instance, the evaluation frequency for higher-levels of an IC, e.g. at CU_A, should be lower than for lower-levels, e.g. $Sys.Res$. [14].

Based on these considerations, we distinguish two main interrelated phases, or states, in the control process (Fig. 4): 1) controller (self-)integration; and 2) system control. Phase (1) represents a *meta controller* (or *meta feedback loop*), for integrating a controller that can reach its goals. In phase (2), this IC controls the underlying system, and is evaluated accordingly. If evaluation is positive, the IC maintains its composition. If negative, phase (1) is performed again (i.e. controller reintegration). Of course, before being evaluated negatively, each CU may also attempt to self-adapt internally (e.g. self-optimisation or configuration) – not shown. If a new IC cannot be found, the integration process halts and waits for a change (e.g. if CUs are missing, wait for new CUs).

In principle, a controller may start in any of the two sates. In practice, it should start in state (2), i.e. pre-integrated controller (developed offline), and self-adapt its composition during its life-cycle. Also, different $(sub-)ICs$ may be in different states, simultaneously – e.g. different ICs pursuing different goals. Important questions must be addressed in future work on the coordination between these two processes.

C. Key Characteristics

Importantly, the proposed (self-)integration process imposes *neither* direct CU communication, *nor* linear goal decomposition (even if this is the case in the current prototype –IV). It merely implies *causality* between required and provided goals: for a CU's provided goals to be fulfilled, its required goals must also be fulfilled, hence provided by other CUs. This allows for higher-level provided goals to 'emerge' from lower-level required/provided goals, meaning, that the exact causality between them is difficult, or impossible, to define analytically.

Controller self-integration is completely *decentralised* – distributed across special-purpose functions implemented within CUs (Cf. IV). The top-down and bottom-up (yoyo) iterations represent a basic decentralised learning process, based on *trial-and-error* searching for a viable ICs – i.e. a CU combination that reaches the goals. Future work may include more sophisticated learning and self-optimisation processes (slower, more 'thoughtful'), running in parallel with these basic self-integration processes (faster, more reactive). Once learned, optimised controllers may be (self-)integrated much faster.

The proposed two-phase controller design is in-line with Ross Ashby's view on *ultrastable* systems [11], which also feature two types of feedback loops. The first one operates frequently and makes small adjustments for keeping the system within its viability space – the equivalent of an operational IC (Phase-2) in our case. The second feedback operates infrequently, only when the system approaches its viability limits, and performs more dramatic changes, or system restructuring – the equivalent of controller re-integration (Phase-1). Ashby also considers trial-and-error as a *"necessary"* means of adaptation in unknown environments, not at least because of its key role in gathering essential information (i.e. learning).

The proposed design achieves *reusability* not only by reusing CUs in various assemblies, but chiefly by *reusing pre-integrated controllers (sub-ICs)* as parts of larger ICs. For instance, supposing that the stakeholder in Fig. 3 requires a new goal G_{New} that can be provided by a control unit CU_{New}, which in turn requires G. CU_{New} can request G from CU_A, and, if successful, the integration process stops – because CU_A is the top of an existing IC, including CU_B and $Sys.Res$.

This is essential for ensuring *scalability*, by reducing the number of cases where controller integration must be performed, from scratch. The more generic the functionality of *sub-ICs*, the higher their reusability in producing new IC variants. Scalability can be further enhanced by limiting the scope of the trial-and-error integration, at different levels, via hierarchically encapsulated structures (holonic design) [14].

IV. Proof-of-concept Design and Implementation

We developed a proof-of-concept prototype focused on the dynamic integration of controllers for a smart-home system. The objective was to illustrate the feasibility of the proposed design via a concrete example. Concretely, we aimed to show how various CUs, deployed at runtime, can be integrated dynamically for adapting the house's control behaviour to changes in stakeholder goals and smart devices; and how existing ICs can be partially reused to form new IC variants.

The prototype is based on a service-oriented component technology – iPOJO[1], based on OSGi[2], in turn based on Java. It supports CU hot-deployment (i.e. addition, update, removal of OSGi *bundles*) onto the smart home platform – iCASA[3] simulator (also based on iPOJO). The actual specification formalisms and acquisition mechanisms for CUs were out of this prototype's scope (future work).

Each CU was implemented as an iPOJO service-oriented component. It provides and requires services (Java interfaces) that represent provided and required goals, respectively. The integration protocol between goal-requesting and goal-providing CUs consists of a $Goal\ Request$ followed by a

[1]https://felix.apache.org/documentation/subprojects/apache-felix-ipojo.html
[2]OSGi Alliance: https://www.osgi.org
[3]iCASA: https://adeleresearchgroup.github.io/iCasa

Goal Reply message. Each provided goal (service interface) implements a *Goal* interface, which defines a $requestGoal(< parameters >)$ method. Parameters include the goal's $values$ and *scope* [6], [5]. The goal's values V represent its viability domain. Its scope includes a resource scope S_R and a time scope S_T, meaning, *where* and *when* to achieve the goal, respectively. E.g., a $requestGoal$ on a $Temperature - Goal$ specifies parameters $V = 19 - 22°C$ (what), $S_R =$ "$kitchen$" (where) and $S_T =$ "$7 - 10pm$" (when).

Stakeholders can request a goal's activation by calling the $requestGoal$ method on the desired goal interface, provided (implemented) by a CU component. The method's *response* indicates whether or not the goal *request* was accepted (i.e. currently boolean, yet more advanced versions may offer explanations, suggestions, or alternatives). Once accepted, the activated goal must resolve its required goals (III-B). It connects to CUs providing these goals (i.e. dynamic service binding provided by iPOJO) and calls $requestGoal$ for these CUs' goals. This results in a recursive chain of $requestGoal$ calls and responses, which, when all requests have been accepted, leads to a fully-integrated and functional controller.

Within the scope of this prototype, we assumed that integrated CUs are always connected via explicit service bindings (i.e. one CU has a reference to the other one, and uses it to communicate). This needs not always be the case (e.g. indirect communication via the environment - 'stigmergy'). We also assumed that CUs aim to cooperate, when possible. More complicated CU relations – e.g. competition, parasitism or indirect stigmergic influence – will be subject to future work.

We designed three types of CU components:

- Goal Manager (GM): *collects* goal requests, *resolves conflicts* among incompatible goal requests, and *dispatches* coherent goals to Goal Pursuers;
- Goal Pursuer (GP): *controls* resources – e.g. monitors, decides and acts – to achieve active goals;
- Resource Adapter (RA): *connects* CUs to heterogeneous system resources (e.g. device drivers).

These types represent logical components and can be implemented in various ways. For instance, the GM can be a stand-alone component (as in our current implementation), or embedded within the membrane that encapsulates GP components (e.g. iPOJO membrane handlers – we assume this to simplify Figs. 5 and 6). Moreover, logical control functions of GMs and/or GPs can be implemented via various design patterns (centralised, decentralised or hierarchical) [15]. Based on this typology, ICs are assemblies of GPs, connected to home devices via RAs. GPs and RAs that receive conflicting goal requests are encapsulated within GM membranes.

V. Smart Home Case Study and Results

A. Simulation settings

We tested our controller prototype for managing a smart home – simulated via iCASA (Fig. 6-a). iCASA offers several types of smart devices – e.g. heaters, thermometers, presence sensors, and lamps – which we extended with windows and blinds. Devices are equipped with sensors and actuators, via which our CUs could manage the smart home.

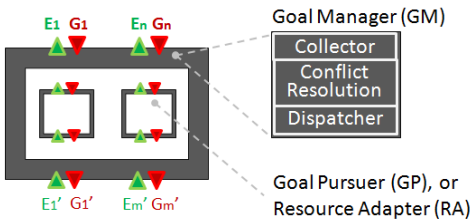

Fig. 5. Components types: Goal Manager (GM – encapsulates control units to resolve goal conflicts), Goal Pursuer (GP – pursues active goals), Resource Adapter (RA – connects controllers to system resources)

We defined the following types of goals: temperature (G_T), energy consumption (G_E), security (G_S) and luminosity (G_L). We implemented various CUs (GMs, GPs and RAs) for integrating controllers able to attain these goals (Fig. 6-b,c,d). Goal types were hot-deployed onto the iCASA simulator progressively, without restarting. For each new goal, the home IC recruited and self-integrated additional CUs, and resolved conflicts with CUs pursuing other goals. Some goals were later removed and/or updated and the IC adapted accordingly. We present a simplified version of these experiments next.

B. Experimental Control Behaviour

Figs 6-b,c,d illustrate the incremental extension of the smart home's IC for handling new user goals. In the first scenario (Fig. 6-b), iCASA platform is started and the user requests a Temperature Goal: $G_T = ([19 - 22°C], kitchen, 7 - 10pm)$. The request is made on a top-level Goal Manager (GM_{IC}), which provides all goals supported by the smart home (G_T, G_E, G_S). GM_{IC} is not shown for simplicity, but we consider it embedded in IC_1's membrane. Once it accepts G_T, GM_{IC} finds a CU that can provide G_T: goal pursuer GP_T. Once GP_T accepts the request for G_T, it searches for i) CUs that provide temperature monitoring G_{MT}: e.g., a thermometer driver RA_T; and ii) CUs that can act on devices that change the temperature: e.g., heater and window adapters, RA_H and RA_W, respectively. To comply with G_T's scope, these CUs must be in the $kitchen$. GP_T also monitors the outside temperature. Since the external temperature is higher than the targeted temperature, GP_T asks RA_W to open the window; it also asks RA_H to switch on the heater. G_T is thus achieved.

In the second scenario (Fig. 6-c), the user further requests a security goal G_S for the entire house, which includes the kitchen. The integration process proceeds as before: GM_{IC} recruits GP_S, which in turn recruits all adapters to devices impacting security – e.g. the window adapter RA_W. When a room is empty, GP_S asks all adapters to *secure* devices and avoid intrusion. For RA_W, this translates into *closing* the window. This causes a *conflict* with the temperature controller GP_T, which had requested the window *open*. The goal manager of the window adapter (GM_W) – not shown, in RA_W's membrane – resolves the conflict by denying the request from GP_T and closing the window (e.g. because G_S

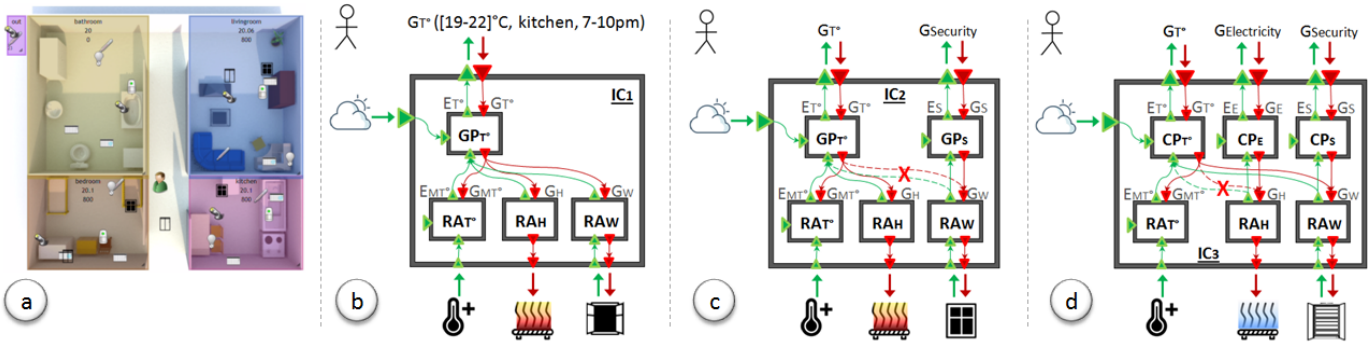

Fig. 6. iCASA smart home simulator (a), with Integrated Controller for Temperature (b), plus Security (c), plus Electricity (d) Goals

has priority over G_T). Hence, GP_T only achieves G_T via the heater device. Both G_T and G_S are attained in this way.

In the third scenario (Fig. 6-d), the user further requests to minimise electricity consumption G_E. As before, GM_{IC} recruits a goal pursuer GP_E, which binds to all adapters of devices that consume energy – e.g. the heater adapter RA_H. During peak hours, GP_E asks device adapters to *switch-off* electric devices. This causes another *conflict* with GP_T, which requested the heater *switched-on*. Because G_E has priority over G_T, the goal manager of the heater adapter (GM_H) – not shown – ignores the requests from GP_T. Hence, GP_T can no longer attain G_T, because its requests to both the window and the heater are turned-down.

Next, the user installs new blinds on the kitchen window, and updates the window adapter (RA'_W) to access its controls. RA'_W can map requests to open the window into several actions: i) open window and blinds; ii) open window and close blinds; and iii) open window, close blinds and tilt them horizontally. RA_W can also map requests to secure the window into the actions ii) and iii) above. Hence, both requests from GP_S and GP_T can now be accepted. Namely, to reach G_S, the window can be open if the blinds are down and tilted; this helps meet G_T since heat from outside can pass through the open window and blinds. Thus, all goals are attained.

C. Discussion

These experimental scenarios showed how ICs could be integrated dynamically from CUs. This enabled ICs to be extended and updated in response to new goal requests and device deployments. CUs were added to the running smart home platform manually; future versions may fetch CUs from special-purpose repositories instead. CU descriptions and implementations were defined and configured specifically for the targeted application. More flexibility and reusability can be achieved in the future by identifying domain-specific control functions and implementing them as reusable CUs.

Most importantly, we showed how *conflicts* could arise dynamically between ICs with incompatible goals. These conflicts were resolved by *merging* the behaviours of the conflicting ICs, rather than by blocking one of them completely. Namely, in the second scenario, the temperature controller GP_T (with lower priority) was not completely deactivated, only its access to the window adapter was blocked. This allowed it to continue functioning and hence to achieve its goal G_T via the heater. In the third scenario, GP_T was again granted access to the window, despite the apparent conflict with the security goal G_S, since the window could be configured to open securely. This approach is similar to the adaptive behavioural merging discussed in subsections II-B and II-C, where partial neural inhibitions enabled the synthesis of new behaviours via reuse of existing neural circuitry.

VI. Related Work

Many research areas are relevant to the proposed approach. We focus here on those relevant to system (self-)integration.

We have already indicated (above) the relevance of service- and component-oriented models (e.g. iPOJO/OSGi, but also CCM, EJB, .NET, Fractal, Web Services) and the key extensions of our proposal. We also discussed the compatibility of our proposal with controller designs in nervous systems (e.g. crayfish) and cybernetics (e.g. Ashby's ultrastable system). Self-integration also resembles self-expression, as in [9], enabling self-adaptive systems to change their collaboration pattern dynamically for dealing with significant context changes.

In the area of self-assembling systems, [7] aim to enable software systems to learn to optimise their composition from alternative building blocks, to maximise performance within each context. The additional complexity in our case is that building blocks are themselves self-* controllers, hence introducing new sources of dynamism to the self-assembly process.

The general area of systems-of-systems [10] is also similar to our objectives. As above, in addition to integrating 'classic' software systems, we aim to (self-)integrate self-* systems.

Recent works aim to tackle conflicts in self-* systems, e.g. [17], [8]. [17] propose an offline service orchestrator that prevents conflicts during service composition. [8] allow conflicting services to be deployed, and enable service developers to program conflict prevention at the service level, based on locking mechanisms offered by the underlying platform.We also aim to handle conflicts that have not been predicted and resolved during the development phase; by detecting them and integrating conflict-related CUs dynamically. In previous work we defined conflicts from a goal-oriented perspective

[16] and proposed specific design patterns for distributing the conflict resolution logic throughout system components [15].

VII. Conclusions and Future Work

This paper focused on two key design features of autonomic controllers – namely, *modularity* and dynamic *(self-)integration* – as key enablers for viable self-* systems, evolving in complex unpredictable environments. The key benefits of these design features include efficiency, scalability and survivability, through reusability (both at elementary and composite levels) and dynamic restructuration (both pre-wired and via trial-and-error learning).

First, the paper brought to the fore three generic controller design features that had been studied in organisms with relatively primitive nervous systems (i.e. crayfish): i) neural configuration: reusing neurons with different output configurations to synthesise various behaviours for different situations; ii) behavioural integration: merging pre-wired neural circuitries and inhibiting conflicting overlaps to produce composite behaviours in complex situations; iii) reafference: using lower-level feedback loops to implement higher-level afferent (control input) signals. Based on a hierarchical organisation of neural circuitry, these features provide an efficient adaptable way of producing a wide variety of behaviours based on a limited amount of resources (sensor, motor and cognitive).

Second, the paper showed how these principles can be adopted for designing complex controllers in artificial self-* systems. Within the broader context of a generic control architecture developed in previous work [14], [5], we focused here on the design features enabling the dynamic (self-)integration of software controllers. We showed how a relatively simple protocol, based on goal requests and replies, could recursively self-integrate controllers from reusable control units (CUs), for responding to unexpected changes (e.g. new goals and devices). This notably included cases where the dynamic integration of pre-integrated controllers (ICs) led to *conflicts*; and how these could be resolved by integrating special-purpose CUs (i.e. conflict-resolution functions). We also showed how ICs, once operational, could be evaluated, and potentially dismantled and reintegrated if they failed to meet their goals.

The resulting (self-)integration/evaluation process follows a top-down and/or bottom-up iterative path, which converges when finding an IC that can 'satisfice' the goals; or when the search space is depleted. Importantly, this approach does *not* assume linearity or reducibility (e.g. via an analytical model) of the behaviour of ICs to the functions of their CUs and to their interconnections. It merely enables a (self-)integration process able to search the combinatorial space of CUs for a suitable composition, in the given context. *Scalability* is a key issue here – we have started to address this in previous work by identifying special-purpose design principles for complexity management in large-scale self-* systems [14], [5].

The proposed design draws inspiration from its natural counterpart by adopting the hierarchical organisation and the associated mechanisms for dynamic reconfiguration, both of individual elements (neurons or CUs) and of pre-integrated elements (neural circuitry or ICs). We argued that this can provide the same advantages to artificial controllers: reusability, efficiency and adaptability to a broad range of changes.

This preliminary work opens several directions for future work – e.g. alternative goal and protocol specifications; CU typology; convergence, stability and scalability of parallel self-integration processes. More experiments are also required to address more complicated scenarios, such as cases where goal requests are denied and several integration iterations are required before convergence.

On the longer term, we envisage adopting more advanced learning functions for self-optimising the self-integration processes in new scenarios and for speeding-up these processes when similar scenarios reoccur. Overall, the presented work aims to contribute towards providing a reusable design model, framework and platform for facilitating the development and maintenance of complex controllers for self-* systems.

References

[1] K. L. Bellman, F. B. Krasne, "Adaptive Complexity of Interactions Between Feeding and Escape in Crayfish", Science, 221-4612, 1983

[2] A. P. Kramer, Franklin B. Krasne, Kirstie L. Bellman, "Different command neurons select different outputs from a shared premotor interneuron of crayfish tail-flip circuitry", Science, 214-4522, 1981

[3] H. Reichert, J. J. Wine, "Coordination of lateral giant and non-giant systems in crayfish escape behavior", Journal of Comparative Physiology, Springer-Verlag 1983, 153:3-15

[4] E. Von Holst, H. Mittelstaedt, "The Principle of Reafference: Interactions Between the Central Nervous System and the Peripheral Organs", 1950,

[5] A. Diaconescu, "Goal-oriented Holonic Systems", in C. Müller-Schloer and S. Tomforde Edts., Organic Computing: Technical Systems for Survival in the Real World, Springer Intl. Pub., Dec. 2017

[6] S. Frey, A. Diaconescu, D. Menga, I. Demeure, "A Holonic Control Architecture for a Heterogeneous Multi-Objective Smart Micro-Grid", Intl. Cnf. on Self-Adaptive and Self-Organizing Systems (SASO), 2013

[7] B. F. Porter, R. Rodrigues Filho,"Losing control: the case for emergent software systems using autonomous assembly, perception and learning", Intl. Cnf. on Self-Adaptive and Self-Organizing Systems (SASO), 2016

[8] R. Ben Hadj, et al. , "Context-based conflict management in pervasive platforms", 2017, PerCom Workshops 2017, pp 250-255.

[9] F. Zambonelli et al., "On Self-Adaptation, Self-Expression, and Self-Awareness in Autonomic Service Component Ensembles", Intl. Cnf Self-Adaptive and Self-Organizing Systems Workshops (SASO-W), 2011.

[10] A. Sage and C. Cuppan, "On the Systems Engineering and Management of Systems of Systems and Federations of Systems", Information-Knowledge-Systems Management Journal. 2(4), 2001.

[11] W. Ross Ashby, "The Ultrastable System" pp 80-99, in "Design for a Brain", 2nd Ed., 1960

[12] C. Escoffier, R. S. Hall, P. Lalanda, "iPOJO: an Extensible Service-Oriented Component Framework", IEEE SCC 2007, pp 474-481

[13] "Quantitative Organic Computing. Controlled Emergence", in C. Müller-Schloer and S. Tomforde Edts., Organic Computing: Technical Systems for Survival in the Real World, Springer Intl. Pub., Dec. 2017

[14] A. Diaconescu, S. Frey, C. Müller-Schloer, J. Pitt, S. Tomforde, "Goal-oriented Holonics for Complex System (Self-)Integration: Concepts and Case Studies", IEEE Intl. Cnf. on Self-Adaptive and Self-Organizing Systems (SASO'16), Augsburg, DE, 2016, pp 100-109

[15] S. Frey, A. Diaconescu and I. Demeure, "Architectural Integration Patterns for Autonomic Management System", Intl. Cnf. on the Engineering of Autonomic and Autonomous Systems (EASe), 2012

[16] S. Frey, A. Diaconescu, D. Menga, I. Demeure, "Towards a reference model for multi-goal, highly-distributed and dynamic autonomic systems", Intl. Cnf. on Autonomic Computing (ICAC), 2013

[17] R. B. Baghli, E. Najm, B. Traverson, "Defining services and service orchestrators acting on shared sensors and actuators", Intl. Cnf. on Model-Driven Engineering and Software Development, 2018

[18] W. T. Powers, "The Reafference Principle and Control Theory", Perceptual Control Theory (PCT), www.pctweb.org/ReafferencePCT.pdf