Chapter 4
Reference Scenarios for Self-Aware Computing

Jeffrey O. Kephart, Martina Maggio, Ada Diaconescu, Holger Giese, Henry Hoffmann, Samuel Kounev, Anne Koziolek, Peter Lewis, Anders Robertsson and Simon Spinner

Jeffrey O. Kephart
Thomas J. Watson Research Center, Hawthorne, NY USA, e-mail: kephart@us.ibm.com

Martina Maggio
Lund University, Department of Automatic Control, Ole Rmers vg 1, SE 223 63 Lund, Sweden, e-mail: martina.maggio@control.lth.se

Ada Diaconescu
Telécom ParisTech, Equipe S3, Departement INFRES, 46 rue Barrault, 75013 Paris, France, e-mail: ada.diaconescu@telecom-paristech.fr

Holger Giese
Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany, e-mail: holger.giese@hpi.uni-potsdam.de

Henry Hoffmann
University of Chicago, Department of Computer Science, Ry 250, Ryerson Hall, 1100 E 58th St, Chicago, IL 60637, e-mail: hankhoffmann@cs.uchicago.edu

Samuel Kounev
University of Wrzburg, Department of Computer Science, Am Hubland, D-97074 Wrzburg, Germany e-mail: samuel.kounev@uni-wuerzburg.de

Anne Koziolek
Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology, Am Fasanengarten 5, D-76131 Karlsruhe, Germany e-mail: koziolek@kit.edu

Peter Lewis
School of Engineering and Applied Science, Aston, University, Birmingham, UK, e-mail: p.lewis@aston.ac.uk

Anders Robertsson
Lund University, Department of Automatic Control, Ole Rmers vg 1, SE 223 63 Lund, Sweden, e-mail: anders.robertsson@control.lth.se

Simon Spinner
University of Wrzburg, Department of Computer Science, Am Hubland, D-97074 Wrzburg, Germany e-mail: simon.spinner@uni-wuerzburg.de

Abstract This chapter defines three reference scenarios to which other chapters may refer for the purpose of motivating and illustrating architectures, techniques and methods consistently throughout the book. The reference scenarios cover a broad set of characteristics and issues that one may encounter in self-aware systems, and represent a range of domains and a variety of scales and levels of complexity. The first scenario focuses on an adaptive sorting algorithm, and exemplifies how a self-aware system may adapt to changes in the data on which it operates, the environment in which it executes, or the requirements or performance criteria to which it manages itself. The second focuses on self-aware multi-agent applications running in a data center environment, allowing issues of collective behavior in cooperative and competitive self-aware systems to come to the fore. The third focuses on a cyber-physical system. It allows us to explore many of the same issues of system-level self-awareness that appear in the second scenario, but in a different context and at a potentially even larger (potentially planetary) scale, when there is no one clear global objective.

4.1 Introduction

This chapter introduces three reference scenarios for which self-awareness plays an important role. The reference scenarios are intended to cover a broad set of characteristics and issues that one may encounter in self-aware systems, and to represent a range of domains and a variety of scales and levels of complexity. They constitute a starting point that will be elaborated further in other chapters, as needed.

The remainder of the chapter is organized as follows. In Section 4.2, we explain the criteria that led us to choose this particular set of reference scenarios, including a set of research questions that we wished to expose, and a set of dimensions that we wished to explore.

The first scenario, presented in Section 4.3, focuses on an adaptive sorting algorithm, and exemplifies how a self-aware individual system element may adapt to changes in the data on which it operates, the environment in which it executes, or the requirements or performance criteria to which it manages itself.

The second scenario (Section 4.4) features applications running in a data center environment. It brings to the fore issues of coordination, cooperation, and competition that arise within self-aware applications composed of multiple interacting self-aware elements or components. Moreover, it raises issues of competition and conflict that may arise among multiple self-aware applications, and between those applications and a self-aware entity that represents the interests of the data center owner.

The third scenario (Section 4.5) focuses on a cyber-physical system. It allows us to study issues of system-level self-awareness similar to those of Section 4.4, but in a different context and at a potentially even larger (potentially planetary) scale, at which there is no one clear global objective. This scenario is built up incrementally in increasing levels of scale and complexity, thereby highlighting different levels of self-awareness. The chapter concludes with a brief summary (Section 4.6).

4.2 Rationale

The purpose of this section is to explain some of the considerations that led to our choice of reference scenarios, including a set of research questions and dimensions that we wished to explore.

Among the key research questions regarding self-aware systems that we wish our reference scenarios to support are:

- How can self-awareness help a computing system achieve its goals?
- What is the relationship between a system's properties and the type and degree of self-awareness that is most appropriate or beneficial for it?
- What are the costs or drawbacks of self-awareness, and what is its overall net benefit?
- How do different instances of self-awareness operating at different levels of a system interact with or otherwise affect one another, and what is the impact of these interactions upon overall system behavior and performance?

Among the key dimensions of self-awareness that we wished to explore through the reference scenarios were:

- **Goal complexity**. For any given self-aware entity, does its goal concern a single attribute (such as a thermostat that just manages temperature), or is it multi-attribute (as in the data center scenario, where an application owner might strive to optimize multiple application performance criteria (involving response time, throughput, and down-time) while minimizing resource usage (so as to reduce payments due to the data center owner)?
- **Goal alignment**. For self-aware systems that comprise more than one self-aware entity, to what degree do the objectives of the constituent entities align with one another? This is related to whether or not the individual entities are operating explicitly on behalf of one authority, or several.
  - The adaptive sorter scenario represents a simple one-component system with a single purpose.
  - The data center scenario exemplifies a multiplicity of individual goals held among the end users, the application owners, and the data center owner.
  - The various cyberphysical scenarios cover a range of cases, including:
    - Appliances that have their own individual objectives but may have some consideration for global house-wide objectives built into them;
    - Smart homes that each seek to minimize cost and maximize power consumption by their owners, potentially creating resource contention and possibly resource shortages; and
    - Shuttles that are designed to be highly cooperative with one another.

- **Heterogeneity**. Self-aware systems may tend towards homogeneity or heterogeneity in terms of technology and protocols as well as behaviour, strategies, self-management capabilities, self-awareness level, degree to which they have adapted effectively to the environment in which they are situated, and timing characteristics [20, 21].

4.3 Adaptive Sorting

What might it mean for software to be adaptive and self-aware? To answer this question, we start by considering as an example the `std::sort` algorithm included in the C++ Standard Template Library (STL). In its current form, the STL sorting algorithm is neither adaptive nor self-aware. However, there are two important reasons why one might want to endow it with adaptability and self-awareness: coping with the plethora of different hardware architectures on which the algorithm might run, and coping with the wide variety of input data characteristics.

First, consider the influence of the hardware architectural features such as cache size, cache line size, and the number of registers on the nature of the optimal sorting algorithm and parameters. In the version of `libstdc++` included with GCC 4.3, merge sort was used until the list was smaller than 15 elements, below which insertion sort was used. This choice was established empirically as best for problems of a certain size and type on architectures that were common when the library was originally written. By 2009, architectures had evolved to the point where a higher cutoff was found to be more effective [1]. [1]

Second, consider the influence of data set characteristics such as size and distribution (e.g. standard deviation) upon the optimal sorting algorithm and parameters. Experiments reveal that data sets with small standard deviations favor the quicksort algorithm, while for larger standard deviations the CC-radix sort [17, 29] is the best choice. When the number of keys is increased, the best algorithms for small standard deviation values are multi-way mergesort, while CC-radix is best for larger input sizes and higher standard deviation [22].

Recently, researchers have begun to experiment with generating sorting algorithms dynamically, based on observations of the performance and the execution platform and fine grained performance tuning [1,4]. This automatic tuning is usually done through an empirical search [18, 19, 26] that identifies the algorithm (among a set of potential ones) that performs best on the specific deployment machine. Code that dynamically adapts to the characteristics of the input has a significant advantage over other types of optimization [22].

Having established why it is advantageous for a sorter to be capable of adapting to its environment (i.e. the hardware architecture and the data upon which it operates), we now imagine what an adaptive, self-aware sorter might be like.

[1] Another notable example where hard-coded parameters have evolved over time to accommodate changes in architecture is the Discrete Fourier Transform [4, 9].

4.3.1 Scenario

An adaptive, *self-aware sorter* should sense the environment and the input data to be sorted and adapt its behavior accordingly. These requirements have both design-time and run-time implications.

First, consider the design of a self-aware sorter. The designer must anticipate that the sorter could be instantiated in a variety of different environments, including different hardware architectures, or different amounts of compute resources such as CPU, memory, and I/O. The designer must also anticipate that at run-time the user of the sorting algorithm could express goals and constraints in terms of a variety of different high-level attributes, including latency, throughput, energy consumption, etc. and that the data set could vary widely in terms of size or other characteristics that might affect sorting efficiency. The designer must then incorporate her knowledge or expectations into the design of the self-aware sorter. The design must include:

1. means for sensing certain aspects of the environment (e.g. the identity of the operating system, available compute resources, or the data-set size);
2. means for controlling certain aspects of the sorter's behavior (e.g. the type of sorting algorithm used, or the amount of memory allocated);
3. means for sensing certain attributes of the self-aware sorter's behavior that might possibly be of interest to the user;
4. knowledge of which environmental aspects and algorithmic behavior attributes can be sensed, and which algorithmic parameters can be controlled;
5. models (or partial models, or hints about the likely functional form of models whose details could be learned at run time) that capture the dependency of behavioral attributes upon environmental conditions;
6. means for capturing user goals and constraints; and
7. means for using models to optimize control parameters with respect to user goals and constraints

The design goals, run-time expectations (and ranges into which user goals fall), and models or model hints should be expressed in a form that can be used by formal verification methods to confirm that the sorter is correct and satisfies the designer's goals and objectives (which include coping with the widest possible set of conditions and user expectations).

Now consider the self-aware sorter at run-time. When instantiated within a specific environment, the self-aware sorter senses the hardware and resources of that environment, assesses the data set that it is being asked to sort, and receives information about the user's goals and constraints with regard to latency, throughput, energy consumption, or other attributes of interest. For example, the user might wish the sorter to minimize CPU and memory usage while maintaining a compute time of no more than 5 seconds. Given the initial model provided by the designer, its understanding of the environment in which it is situated, its own state and capabilities, and the user's objectives, the self-aware sorter then uses its optimization capability to determine the best sorting algorithm and associated parameters to use, and perform the sort accordingly.

In a more advanced variant of the scenario, the self-aware sorter dynamically searches for new sorting algorithms not contained within its original code base that are more optimal given current circumstances. Such algorithms might exist in the form of libraries that could be dynamically linked into the code, or a web service. In order to determine that these new algorithms are likely to be better than any currently used by the sorter, new algorithms should be accompanied by models that predict their likely performance. Such models might be packaged with the algorithms themselves, for example, or provided by a trusted third party that conducts extensive experiments on the algorithms in order to learn models for them.

In an orthogonal variant, models relating environmental conditions and algorithmic parameters to high-level performance criteria might be learned dynamically as the sorter is applied to various data sets over time. If the designer has already provided a model, the model learning might take the form of Bayesian updates to the original model. If the designer has instead provided hints about relevant variables and/or some expectation about the structure of the model, these could be used as a framework or constraint in which the learning would take place. In any case, the learning could be performed by the sorter itself, or it could be done by third parties that do the learning and then make the resultant models available.

Preferably, the design specification should be accessible by the instantiated self-aware sorter. If for some reason the assumptions under which the design was performed and verified are violated, an alert could be created and sent to the designer or to the verification algorithm. Upon learning that the original assumptions are violated, the designer would have a chance to re-design the algorithm, or the verification algorithm could be launched automatically and signal the designer if the run-time conditions are such that the algorithm is not sound (or optimal). Moreover, other properties can be taken into account, like software health [28].

4.3.2 Key Questions

- How feasible is it for the designer of a self-aware software component to build in the required sensors, controls, models, and self-regulation mechanisms? Must we invent a new generation of software design tools to help with this task?
- What type of information could a self-aware software component capture at runtime that would support improvements in the design of the next generation of that software? What are methods by which such information captured from multiple runtime instances of self-aware sorters be integrated into the redesign?
- Is it possible to remove the human designer from the loop, such that the redesign step becomes a type of runtime adaptation? How would this work?

4.4 Data Center Resource Management

As a second reference scenario, we consider self-awareness in the context of a data center. This is an interesting case because data centers involve management of thousands of applications, and are therefore several orders of magnitude beyond the self-aware sorter in terms of scale and complexity. Moreover, in this case there are many different stakeholders instead of just one:

- the *data center owner*, who operates the physical infrastructure, consisting of a large number of physical (and, increasingly, virtual) machines plus network and other computational resources required to run applications. The data center owner seeks to honor its service level agreements with the application owners while minimizing the costs of building and maintaining the data center.
- multiple *application owners*, who purchase compute resources from the data center owner and use those resources to deploy, run and manage applications that provide services to end users. They seek to honor their SLAs with end users while minimizing what they pay to the data center owner for use of the physical infrastructure; and
- myriad *end users* who use the applications provided by the application owners and are concerned only with their own perceived service quality. Web users typically tolerate a waiting-time between 2 and 4 seconds [25], depending upon the type of application (e.g., interactive web page vs. mail delivery service). There is some price elasticity, i.e. end users may be willing to pay more for better service.

The data center owner, application owners and end users may come from different organizations (e.g., in a public cloud) and the usage of the service and the infrastructure may be connected with certain fees. The expectations regarding the quality of service and the fees are defined through service-level agreements (SLAs) between the data center owner and the application owners, as well as SLAs between application owners and their end users.

The need for data centers and the applications that run in them to be adaptive is well-recognized. Many applications are subject to strongly time-varying workloads that can (during so-called “flash crowds”) reach peaks five times more intense than the average workload [5], causing the resource requirements to increase accordingly [27]. Moreover, unexpected hardware failures occur frequently in data centers [14, 24].

Virtualization is a key technology that has been introduced to enable more dynamic management of resources and applications in data centers. Virtualization flexibly allocates computing, storage and network resources to applications running in data centers. Data center owners benefit because they can consolidate independent applications onto the same physical hardware, thereby reducing the physical resource and electrical power required to support a given number of applications. Virtualization allows a data center owner to overcommit physical resources, i.e. they can allow the allocated virtual machine (VM) resource reservations to exceed the current physical resource capacity, on the assumption that not all virtual machines require the resources at the same time [2, 15]. Application owners benefit because they can dynamically acquire additional computational resources whenever their application workload increases, and release those resources back to the data center when it decreases — which is much more efficient and cost-effective than the traditional method of requesting enough dedicated physical resource to satisfy the maximum anticipated demand.

Virtualization enables efficiency and cost-effectiveness for both data center owners and application owners, resulting in savings that can be passed on to the end users. In practice, however, it is challenging to manage virtualization in such a way that objectives are met fairly and efficiently as workloads fluctuate, and goals shift.

One factor that contributes to this challenge is that modern virtualization and middleware platforms provide a wide range of adaptable parameters, such as where virtual machines are placed on physical hosts, the number of VM instances, the size of each VM (e.g., number of virtual CPUs), the scheduling priorities for a VM (e.g., CPU reservations, limits and shares), and the platform configuration (e.g., thread pools, cache size).

Another factor is that it is difficult to know how changes to the parameters will affect the behavior of the system, and its impact upon the various stakeholders. Typical cloud management solutions of today (such as Amazon EC2[2], or CloudStack[3]) use auto-scaling techniques to add or remove resources from an application when low-level metrics such as CPU utilization cross a given threshold — a very rough proxy for the metrics of actual interest to application owners and end users, such as application performance and robustness. These VM management solutions are not only unaware of the high-level goals of the stakeholders; they would not know how to manage to those goals because they lack models that map from VM management parameters to application-level metrics.

A final factor is that today's VM management systems do not take into account the multiple conflicting goals of the various stakeholders in a principled way. Indeed, the interests of the data center owner and the application owners are often completely at odds with one another; for example, data center owners wish to minimize costs by minimizing use of power and physical resources, while application owners want the higher performance that results from maximizing power and physical resources. Consequently, VM management systems are unlikely to realize tradeoffs that are understandable or fair. For example, some modern virtualization techniques attempt to optimize resource allocation at the data center level. The VMware Distributed Resource Scheduler (DRS) [13] balances and distributes the load between physical hosts in a data center by migrating virtual machines. The Distributed Power Management (DPM) controller [13] automatically consolidates VMs if physical hosts are underutilized, placing freed-up hosts in standby mode to save energy and rebooting when necessary. While an effort is made to minimize the likelihood that an application will be starved for resource while waiting for a reboot, these controllers do not consider application performance requirements explicitly.

[2] `http://aws.amazon.com/en/ec2/`

[3] `http://cloudstack.apache.org/`

4.4.1 Scenario

Here we exemplify how self-awareness could enable more effective use of virtualization from the perspective of the two stakeholders who have the ability to control resources: the application owner and the data center owner.

A hypothetical self-aware controller operating on behalf of the application owner [31] wishes to use an appropriate amount of resource, such that the fees it pays to the data center owner for the use of physical or virtual compute resources are no more than they need to be to satisfy the demand from their end users. To do this, an application controller could take two types of actions. First, it requests virtual computational resources $\mathbf{r}$ from the data center owner (at some cost), and second, it sets the values of the parameters under its control $\mathbf{c}(\mathbf{r})$ to allocate and control the virtual resource $\mathbf{r}$. The virtual resource might for example be a week of time on a virtual machine with a CPU capacity of $5 \cdot 10^9$ cycles per second, with 8GB of memory and 80GB of SSD storage.

The application controller's goal is to maximize its revenue from the end-user SLAs, $\pi(\mathbf{V})$, where $\mathbf{V}$ represents the values of the attributes appearing in the SLA. The end-user SLAs would be based upon application-level metrics, such as average response time (or 95% response-time percentiles), latency, and might include penalties for service disruption. Additional application-level attributes might include elasticity [16] (i.e., the ability to rapidly increase or decrease the allocated resource in response to demand from end users) and resource stability (guarantees that fluctuations in the compute resource provided to the application will be minimized[4] .

The self-aware controller has a self-model $\mathbf{V}(\mathbf{c}(\mathbf{r}),\lambda)$ that expresses how its performance attribute values depend upon the control settings and the workload λ. Some methods by which such a model might be learned are discussed in Chapter 12. The self-aware controller then optimizes over all control settings to identify $\mathbf{c}^*(\mathbf{r})$, the setting for which $\pi^*(\mathbf{r}) = \pi(\mathbf{V}(\mathbf{c}^*(\mathbf{r}),\lambda))$ is greatest.

The self-aware controller uses the above procedure to compute $\pi^*(\mathbf{r})$ for all possible values of $\mathbf{r}$. Then, it performs a second optimization over all values of $\mathbf{r}$ to determine the value $\mathbf{r}^*$ that optimizes the net profit $\pi^*(\mathbf{r}) - \gamma(\mathbf{r})$ (the revenue from the end-user SLAs minus the payment to the data center owner for the resources $\mathbf{r}$). It requests compute resources $\mathbf{r}^*$ from the data center owner, and then when it receives them it allocates and controls those resources according to the optimal control values $\mathbf{c}^*$. Since the model contains a workload-dependent term, it and any optimizations based upon it must be continually updated as the workload fluctuates.

A hypothetical self-aware controller operating on behalf of the data center owner could employ a similar approach involving self-models coupled with optimization, balancing its need to satisfy SLAs that it has in place with each of the application owners against its desire to minimize its own costs for physical infrastructure and power. Specifically, the data center controller can select a set of control parameters

[4] The data center owner might minimize such fluctations by using traditional performance isolation techniques such as physically isolating the compute resources. In contrast to current practice, in which SLAs explicitly mention physical isolation, in our opinion the SLA should be expressed solely in terms of a service guarantee, and not in terms of how that guarantee is implemented.

$\mathbf{c}$ that (according to a self-model $\mathbf{r}(\mathbf{c})$) is expected to produce an amount of virtual resource $\mathbf{r}$. Additionally, the choice of control parameters $\mathbf{c}$ implies an increase $\rho(\mathbf{c})$ in the amount of physical resource and power that must be provisioned in order to realize those control parameters. When an application controller requests an amount of virtual resource $\mathbf{r}'$, the data center controller's task is then to find the $\mathbf{c}^*$ that minimizes the effective amortized cost for physical resources and power $p(\rho(\mathbf{c}))$, subject to the constraint that $\mathbf{r}(\mathbf{c}) = \mathbf{r}'$. The data center will then receive a profit of $\gamma(\mathbf{r}') - p(\rho(\mathbf{c}^*))$ from that application owner.

This scenario is somewhat naïve in that it assumes that resource and control settings are instantaneously responsive. This is not true in general; for example, it can take seconds, or even minutes, to allocate new VMs to an existing application, particularly if a physical server needs to be powered on. In a more sophisticated variant of the above scenario, the self-aware application controller would take into account these time lags, and compute its resource request proactively by anticipating that more resource is likely to be needed soon. Reinforcement learning approaches have proven effective in such cases [30].

In yet another variant of the scenario, the agreement between the application owner and the data center owner would be based upon SLAs describing the virtual resource $\mathbf{r}$ provided to the application controller, rather than directly in terms of the resource itself. The attributes appearing in such an SLA would include traditional resource-level metrics such as CPU cycles, memory allocation, and bandwidth, but it would also include metrics describing service disruptions (e.g. downtime) as well as bonuses and/or penalties incurred when the provided resource is more or less than a specified target amount.

In another variant, the data center controller enforces an overall constraint on the amount of physical infrastructure and power consumed across all applications. This necessitates significant changes to the interaction between the data center owner and the application owners. Since the application owners do not decide unilaterally how much resource they will receive, a negotiation process would now be required.

4.4.2 Key Questions

- Which methods are most efficient and effective for learning models that map from controllable parameters and environmental conditions to SLA attributes? Some answers to this question will be provided in Chapter 12.
- In the scenario above, both controllers used a relatively simple predictive model control approach to govern their actions. More sophisticated approaches are needed to handle situations where there is a delay between when an action is taken and its effect is manifest, such as reinforcement learning. What are the best techniques for exploiting models to select actions, and under what conditions?
- In the scenario above, the data center owner provided to each application owner the amount of resource that they requested, and the application owners then did

their best to manage within that allocation. One can envision other scenarios in which the data center owner determines the amount of resource, or there is some information exchange or negotiation through which both parties can jointly determine the allocation. What other schemes exist, and how well do they work under various circumstances, from the standpoint of both parties? Some answers to this question can be found in Walsh et al. [31], and in Chapter 13.
- For both the data center owner and the application owners, the terms of the SLA have a profound impact on the behavior of the system. What are some means by which these SLAs can be established in the first place? Must they always be set unilaterally, by the data center owner? Or might there be a process of negotiation between the data center owner and the application owners, and if so what are some plausible (or perhaps optimal) negotiation mechanisms?
- Would it be feasible to replace SLAs with an auction or other type of dynamic economic mechanism, and if so how would this affect the nature of the algorithms employed by the controllers?

4.5 Cyber-Physical Systems

In this section, we present several scenarios involving cyber-physical systems in which multiple self-aware entities interact within a shared environment. These scenarios allow us to study individual and collective self-awareness, and the relationships between them. As in Frey et al. [8] and Chen et al. [7], the scenarios are organised in order of increasing complexity and scale.

4.5.1 Thermostat

4.5.1.1 Scenario

The first and simplest scenario takes place within a home and involves a single device pursuing a single goal. Specifically, a thermostat strives to maintain a room's temperature within one degree of a value specified by one of the home's inhabitants. The amount of heating or cooling power that must be supplied to the room depends upon the ambient outside temperature, which can vary over the course of a day or a season. Accordingly, the thermostat is equipped with a thermometer that senses the room's temperature, and it uses a simple model to convert the difference between the observed and desired temperature into a signal that controls an actuator that turns the room heating or air conditioning on or off.

In a more complex variant of the scenario, the thermostat pursues multiple goals simultaneously. For instance, in addition to the temperature goal, the thermostat can be constrained by a power consumption goal, such as a maximum power that may be used to heat or cool the house during a day. Alternatively, a power goal could be defined dynamically by an automatic controller of a local power grid, to which the thermostat is connected. Rather than being defined in terms of a daily threshold, the goal might require that the connected device prioritises or avoids power consumption. The goal may change at regular intervals depending on the overall state of the power grid — i.e., the current balance between power consumption and production. In many cases, the thermostat's temperature and power goals may conflict with one another, as reaching a higher temperature with respect to the external environment requires the thermostat to consume (switch on) power while the power goal may recommend the opposite (switch off). In more complicated cases, an additional (and potentially conflicting) goal may be given to the thermostat in order to minimise the cost of power consumption for the home owner, while the grid controller could impose fluctuating power prices.

In an orthogonal variant of the scenario, the self-aware thermostat is capable of explaining its state, behaviour and plans. Specifically, if queried, the thermostat would be able to report its actions (past, current and planned) and the motivations behind them, evaluate its performance with respect to its goal(s), and perhaps even diagnose failures to meet its goals. Such a degree of self-awareness is key to the ability of the thermostat to improve itself, either dynamically at runtime or by collecting observations that would aid in designing the next-generation self-aware thermostat.

4.5.1.2 Key Questions

- How would extending the thermostat's awareness to encompass a broader context, including the presence of humans or the temperature of adjoining rooms, improve its performance and efficiency? How costly would it be to add the requisite sensors, or communication capabilities, or computational power? Would the extra self-awareness warrant this cost?
- How would extending the thermostat's awareness to encompass a broader set of models, such as daily or seasonal external temperatures, models of other rooms in the house, or power price predictions improve its performance or efficiency? How would the costs of such improvements compare to the benefits?
- What are the best means for resolving conflicts among multiple goals?

4.5.2 Smart Home

4.5.2.1 Scenario

This scenario adds to the previous scenario an additional "smart" window shutter and a washing machine, thereby illustrating a case in which several heterogeneous devices pursue different goals within a single environment, under a single authority. Acting upon signals from sensors that sense sunlight and external and internal temperature, and possibly human presence sensors as well, the smart window shutter pursues goals related to the home's lighting and temperature by controlling actuators that rotate the shutter panels individually. In doing so, the smart window controller simultaneously affects both the temperature and the light intensity within the house.

The self-aware washing machine offers different programmes with different on-off cycles and durations, which can be selected by users to run within a specified timeframe. When connected to a smart grid, the washing machine may avoid power consumption or price peaks by delaying the washing cycle automatically. Note that, since the shutter's actions affect temperature and the washing machine's affect power, the introduction of these two self-aware entities into the environment create the potential for goal conflicts that extend across these devices.

In a more complex version of the scenario, additional thermostats or window shutters are introduced throughout the house, and the user specifies house-wide goals or constraints, such as house temperature and lighting preferences, or power consumption thresholds or costs — thereby requiring cross-device coordination.

Another interesting twist on the scenario considers the consequences of altering the set of sensors or devices installed within the smart home. Individual self-aware devices must adjust to the presence of new sensors or devices that consume power or produce heat, or to the absence of recently-removed sensors or devices. Moreover, any models or controllers that operate at the level of the smart home as a whole must adjust to changes in power-consumption or power-provision profiles or other characteristics of newly-introduced or recently-removed devices.

4.5.2.2 Key Questions

- What conflicts might arise as multiple homogeneous or heterogeneous self-aware devices operating within a single environment attempt to satisfy their individual (and possibly multifarious) goals? For example, how could thermostats cooperate with shutters to manage temperature goals, while simultaneously attending to their respective power and light-intensity goals? How effectively can various negotiation or other mechanisms resolve such conflicts?
- How can a collective of self-aware entities manage to micro and macro goals simultaneously, such as the power consumption of each device and of the house overall? What are the relative merits of a single central self-aware controller for the home as a whole vs. an arbiter or a fully-decentralized arrangement in which global goals are somehow translated into local goals that are combined with the intrinsic goals of the individual controllers?
- How can self-aware entities detect and cope with changes in their environment, such as the addition or removal of sensors or devices?
- How could a self-aware collective (such as a smart home) provide coherent explanations of its behaviour and plans, both at an individual and a collective level?

4.5.3 Smart Micro-Grid

4.5.3.1 Scenario

Consider a scenario in which several smart homes are interconnected via a smart micro-grid. Not only does this move up a level in scale from the previous scenario; it allows one to explore situations in which there is no single authority, but rather multiple authorities with interests that may both overlap and conflict. The interests of each smart home owner may be represented by its own independent self-aware controller, while the interest of the power company that owns the micro-grid may also be represented by its own self-aware controller. The interests of these various authorities overlap in some regards; for example they all desire power service dependability and sustainability. However, they also conflict in other aspects; for example, each home owner desires minimal costs, while the power company wants maximal profits.

Each self-aware controller may act independently of the other, and may even choose to leave the power grid at any time. The power grid authority likely plays a role in determining the power consumed by each smart home, but it may be indirect in nature, or shared with the smart homes (e.g. if the allocation depends upon pricing or some type of negotiation). However, it does not have fine-grained over the allocation of power within individual homes; that is the province of the individual smart home controllers.

Other minor variants of the scenario introduce additional entities representing the interests of other independent authorities, such as

- alternative energy generators (e.g., wind turbines and solar plants);
- governmental or other regulatory authorities that may impose general laws (and hence constraints) directly (e.g., restrictions on power consumption by individual homes) or indirectly (e.g., limits on prices charged by the power company);
- entities other than smart homes that consume electric power: businesses, electric cars, etc., and whose behavior are therefore coupled in myriad subtle and unanticipated ways with that of the smart homes (see for example [23] on the topic of probabilistic modeling, and Chapter 13).

4.5.3.2 Key Questions

- When multiple self-aware controllers with independent objectives interact in the absence of a central authority, what forms of communication and negotiation most ensure that there is sufficient mutual benefit to warrant the interaction?
- In systems of multiple independent self-aware entities, the learning and adaptation of one such entity induces a behavioral change that affects the experience of those with which it interacts. How is it possible for self-aware entities to learn and adapt in an environment that is always changing due to the learning and adaptation of the other self-aware entities?

- In systems of multiple independent self-aware entities, to what extent is it possible or desirable for self-aware entities to become aware of one anothers' goals, internal states and behaviours?
- Given possible restrictions on information access (due e.g. to privacy concerns), what mechanisms and incentives support exchange of useful information among self-aware entities? How might they best employ such information?

4.5.4 System of Autonomous Shuttles

4.5.4.1 Scenarios

As a final cyber-physical systems scenario, consider an intelligent transportation system in which trains of connected railroad cars are replaced by a collection of autonomous shuttles.[5] By operating on-demand, autonomous shuttles can be more efficient, and tailor their operation more precisely to passenger needs and priorities. For example, managers may request high speed transport to an urgent meeting, while tourists may request transport within a certain time window for less cost.

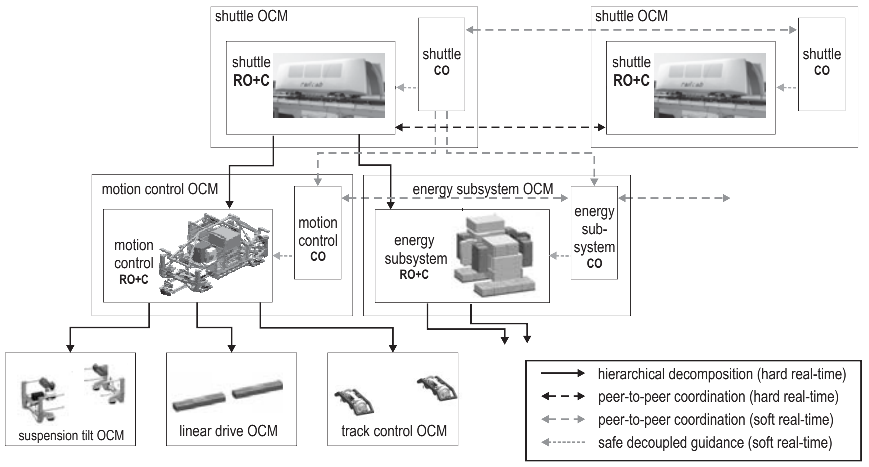

Fig. 4.1: Architecture of the autonomous shuttle as well as their combination (figure taken from [11]).

A prototype autonomous shuttle system of this nature is under development [11]. In order to separate the hard real-time processing required for reliable switching and reconfiguration from the soft real-time processing required for long term decision making, the system features an operator controller module (OCM) containing an

[5] `http://www.railcab.de/index.php?id=2\&L=1`

arbiter that controls the underlying physical processes. The reflective operator handles the necessary reconfiguration in hard real-time. At the same time, a cognitive operator captures the more demanding aspects of self-optimization such as decision making based on run-time models. As depicted in Figure 4.1, the architecture of a single autonomous shuttle is a hierarchy of OCMs[6]. Optionally, the highest-level OCMs within each autonomous shuttle may interact with one another.

In one class of variants, the shuttles operate completely independently, optimizing their own operation without directly sharing any information other than what can be obtained from observation, i.e. they have no awareness of one another's models or intent. For example, they might parastically save energy by following other shuttles when possible to reduce air resistance. In another variant, shuttles compete with another to earn money by transporting passengers or cargo. If the mechanism used to match shuttles with transport requests is an auction, then to form a competent bid the shuttles need to generate good estimates of the time, energy and expense incurred in meeting specific transport requests, which requires that they learn and/or use models of their own capabilities and costs as well as a model of the environment in which they operate. Their bids may be placed with or without an awareness of the existence of other shuttles that are vying to serve the same transport requests.

In another class of variants, the shuttles are aware of one another in the sense of possessing or being capable of learning models of other shuttles. Such a degree of awareness could be advantageous, as it affords the possibility of anticipating their likely bids or operational behaviors.

Another dimension to the autonomous shuttle fleet scenario is whether the shuttles coordinate their actions with one another, and if so whether that coordination is achieved through a central coordinator or in a decentralized fashion via messages exchanged among the shuttles (in which case coordination would be an emergent effect). A natural motivation for such coordination would be to optimize transport orders globally across the entire fleet, maximizing its revenue by optimizing quality of service and minimizing its cost by determining which shuttles are in a position to provide the requested service most efficiently. While fleet-level planning would be conducted across the set of shuttles, each shuttle's individual controller would still be responsible for executing the plan, based on its model of its own capabilities along with characteristics of the environment in which it operates.

In the case where shuttles cooperate by explicitly exchanging high-level information with one another, all of the activities of self-aware computing systems discussed by Weyns et al. [32] may be realized, including

- **Learning run-time models jointly.** Consider a fleet of shuttles that cooperates by exchanging monitoring data to learn the characteristics of other shuttles, the characteristics of the environment, or the characteristics of the shuttle fleets.

[6] The design of a single autonomous shuttle would be an interesting scenario in itself. See the Mechatronic UML approach [6] for the model-driven development of self-optimizing embedded real-time systems, which includes a notion of UML components for hybrid behavior, real-time statecharts extending UML state machines, and the required tool support [6, 10] and analysis techniques [3, 10–12] developed to be able to design safe autonomous shuttles and their internal hierarchical structure with self-optimization.

For example, exchanging data on track conditions [6] can help shuttles optimize their travel to any part of the system that has been been experienced by at least one shuttle.

- **Sharing goals and joint reasoning**. If shuttles exchange their goals, their planning can take into account what other shuttles are likely to do, and might through bilateral or multilateral negotiation arrive at a mutually satisfactory itinerary. For example, they may avoid operating at cross purposes, e.g. bidding against one another or planning conflicting itineraries.
- **Joint actions**. If shuttles share with one another their plans, they have a chance to detect problems or conflicts and renegotiate or discover a solution that will be more globally beneficial. For example, if a shuttle is no longer able to move on its own, another shuttle may be able to help it by pushing it to the next station.

Also of interest is a competitive scenario that inherits the characteristics of the cooperative shuttle scenario presented above, including the autonomy of individual shuttles and the centralized or decentralized coordination that optimizes plans and schedules over an individual fleet, and imbeds that scenario in a larger one in which fleets compete with one another to serve transport requests. This scenario shares much in common with the scenario in which individual shuttles compete with one another, except that now the competition is at the level of entire fleets rather than shuttles. For example, in the case it becomes interesting to consider the potential advantage that one fleet might gain by learning models of competing fleets.

Finally, the scenario may be extended to a larger-scale and more heterogeneous intelligent transport system that encompasses automobiles, taxis, bicycles and other forms of transportation that are represented by self-aware controllers — a diverse milieu that brings together myriad diverse interests and applications (e.g. bus and trucking companies, individual and taxi drivers, traffic management, and city government).

4.5.4.2 Key Questions

- How can a system of autonomous shuttles or other self-aware systems fulfill dependability requirements?
- What coordination mechanisms can permit self-aware systems to satisfy their own individual objectives while functioning competently in a system-of-systems that contains myriad other self-aware systems that represent multiple diverse interests?
- What forms of information can usefully be shared among self-interested self-aware entities in cooperative settings, or competitive settings, or combinations thereof? Depending upon the type of information shared, how can self-aware entities best avail themselves of such information for making individual decision, or for engaging in negotiation or cooperative planning with other self-aware entities? Under what conditions are the cost and effort worthwhile?

4.6 Conclusion

This chapter presented three broad scenarios, providing some insights on the role of self-awareness in the design of smarter systems. The first scenario, an adaptive sorting algorithm, showed a single agent adaptation system, where one entity can exploit its knowledge of the system to improve its behavior. The second scenario showed multiple entities in a data center that could benefit from self-awareness to pursue higher level goals. In this case, there is at least a specific data center owner, together with other playes. The owner and the players goals should be fulfilled. The third set of scenarios showed mulitple entities that can have conflicting goals, where the owner of the system is unclear or undefined. For all these scenarios, the chapter pointed out challenges and opportunities to be explored.

Acknowledgements This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures", and through the LCCC Linnaeus and ELLIIT Excellence Centers.

References

1. Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. Petabricks: A language and compiler for algorithmic choice. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 38–49, New York, NY, USA, 2009. ACM.
2. Luiz Andre Barroso and Urs Hlzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.
3. Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proc. of the 28th Intl. Conf. on Software Engineering (ICSE), Shanghai, China*. ACM, 2006.
4. Anthony Blake and Matt Hunter. Dynamically generating FFT code. *J. Signal Process. Syst.*, 76(3):275–281, September 2014.
5. Peter Bodik, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *SOCC*, pages 241–252, New York, NY, USA, 2010. ACM.
6. Sven Burmester, Holger Giese, Eckehard Münch, Oliver Oberschelp, Florian Klein, and Peter Scheideler. Tool Support for the Design of Self-Optimizing Mechatronic Multi-Agent Systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(3):207–222, June 2008.
7. Tao Chen, Funmilade Faniyi, Rami Bahsoon, Peter R. Lewis, Xin Yao, Leandro L. Minku, and Lukas Esterle. The handbook of engineering self-aware and self-expressive systems. *CoRR*, abs/1409.1793, 2014.
8. Sylvain Frey, François Huguet, Cédric Mivielle, David Menga, Ada Diaconescu, and Isabelle M. Demeure. Scenarios for an autonomic micro smart grid. In *SMARTGREENS 2012 - Proceedings of the 1st International Conference on Smart Grids and Green IT Systems, Porto, Portugal, 19 - 20 April, 2012*, pages 137–140, 2012.
9. M. Frigo and S.G. Johnson. FFTW: an adaptive software architecture for the FFT. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384 vol.3, May 1998.

10. Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*. ACM, 2004.
11. Holger Giese and Wilhelm Schäfer. Model-Driven Development of Safe Self-Optimizing Mechatronic Systems with MechatronicUML. In Javier Camara, Rogério de Lemos, Carlo Ghezzi, and Antnia Lopes, editors, *Assurances for Self-Adaptive Systems*, volume 7740 of *Lecture Notes in Computer Science (LNCS)*, pages 152–186. Springer, January 2013.
12. Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäfer, and Stephan Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the 9th european software engineering conference held jointly with 11th ACM SIGSOFT intl. symposium on foundations of software engineering (ESEC/FSE-11)*. ACM, 2003.
13. Ajay Gulati, Anne Holler, Minwen Ji, Ganesha Shanmuganathan, Carl Waldspurger, and Xiaoyun Zhu. *VMware Distributed Resource Management: Design, Implementation and Lessons Learned*. Mar 2012.
14. Zhenyu Guo, Sean McDirmid, Mao Yang, Li Zhuang, Pu Zhang, Yingwei Luo, Tom Bergan, Peter Bodik, Madan Musuvathi, Zheng Zhang, and Lidong Zhou. Failure recovery: when the cure is worse than the disease. In *HotOS*, pages 8–14, Berkeley, CA, USA, 2013. USENIX Association.
15. James Hamilton. On designing and deploying internet-scale services. In *LISA*, pages 18:1–18:12. USENIX Association, 2007.
16. Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *ICAC*, 2013.
17. D. Jimenez-Gonzalez, J.J. Navarro, and J.-L. Larriba-Pey. Cc-radix: a cache conscious sorting based on radix sort. In *Parallel, Distributed and Network-Based Processing, 2003. Proceedings. Eleventh Euromicro Conference on*, pages 101–108, Feb 2003.
18. T. Kisuki, P. M. W. Knijnenburg, and M. F. P. O'Boyle. Combined selection of tile sizes and unroll factors using iterative compilation. In *Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques*, PACT '00, pages 237–, Washington, DC, USA, 2000. IEEE Computer Society.
19. P. M. W. Knijnenburg, T. Kisuki, K. Gallivan, and M. F. P. O'Boyle. The effect of cache models on iterative compilation for combined tiling and unrolling: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(2-3):247–270, January 2004.
20. Peter R. Lewis, Lukas Esterle, Arjun Chandra, Bernhard Rinner, Jim Torresen, and Xin Yao. Static, dynamic and adaptive heterogeneity in distributed smart camera networks. *TAAS*, 2015 (to appear).
21. Peter R. Lewis, Harry Goldingay, and Vivek Nallur. It's good to be different: Diversity, heterogeneity, and dynamics in collective systems. In *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014*, pages 84–89, 2014.
22. Xiaoming Li, María Jesús Garzarán, and David Padua. A dynamically tuned sorting library. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 111–, Washington, DC, USA, 2004. IEEE Computer Society.
23. O.J. Mengshoel, M. Chavira, K. Cascio, S. Poll, A. Darwiche, and S. Uckun. Probabilistic model-based diagnosis: An electrical power system case study. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(5):874–885, Sept 2010.
24. Meiyappan Nagappan, Aaron Peeler, and Mladen Vouk. Modeling cloud failure data: a case study of the virtual computing lab. In *SECLOUD*, pages 8–14, New York, NY, USA, 2011. ACM.
25. Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour and Information Technology*, 23(3):153–163, 2004.
26. Shah Faizur Rahman, Jichi Guo, and Qing Yi. Automated empirical tuning of scientific codes for performance and power consumption. In *Proceedings of the 6th International Conference*

on High Performance and Embedded Architectures and Compilers, HiPEAC '11, pages 107–116, New York, NY, USA, 2011. ACM.

27. Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *SOCC*, 2012.
28. Johann Schumann, Timmy Mbaya, Ole Mengshoel, Knot Pipatsrisawat, Ashok Srivastava, Arthur Choi, and Adnan Darwiche. Software health management with bayesian networks. *Innov. Syst. Softw. Eng.*, 9(4):271–292, December 2013.
29. Ranjan Sinha and Justin Zobel. Cache-conscious sorting of large sets of strings with dynamic tries. *J. Exp. Algorithmics*, 9, December 2004.
30. Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE, 2006.
31. William E Walsh, Gerald Tesauro, Jeffrey O Kephart, and Rajarshi Das. Utility functions in autonomic systems. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 70–77. IEEE, 2004.
32. Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl Goeschka. On Patterns for Decentralized Control in Self-Adaptive Systems. In Rogério de Lemos, Holger Giese, Hausi Müller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science (LNCS)*, pages 76–107. Springer, January 2013.