Self-Adaptive Software with Decentralised Control Loops

Radu Calinescu[1], Simos Gerasimou[1], and Alec Banks[2]

[1] Department of Computer Science, University of York, UK
[2] Defence Science and Technology Laboratory, Ministry of Defence, UK

Abstract. We present DECIDE, a rigorous approach to decentralising the control loops of distributed self-adaptive software used in mission-critical applications. DECIDE uses quantitative verification at runtime, first to agree individual component contributions to meeting system-level quality-of-service requirements, and then to ensure that components achieve their agreed contributions in the presence of changes and failures. All verification operations are carried out locally, using component-level models, and communication between components is infrequent. We illustrate the application of DECIDE and show its effectiveness using a case study from the unmanned underwater vehicle domain.

1 INTRODUCTION

A growing number of mission-critical software systems operate in uncertain scenarios characterised by internal failures and environment changes. These systems must *self-adapt* to comply with strict dependability, performance and other quality-of-service (QoS) requirements. Achieving QoS compliance is a great challenge of *self-adaptive software* [21]. To address it, recent research proposed the use of formal methods to drive self-adaptation within software systems. A promising approach in this area is *runtime quantitative verification* (RQV) [3, 6, 9], which uses quantitative model checking to reverify the QoS properties of software after environment or internal changes. This reverification identifies, and may in certain scenarios predict, QoS requirement violations, and supports the dynamic reconfiguration of the software for recovery from, or prevention of, such violations [3]. RQV has been successfully used to develop centralised-control self-adaptive software in domains including dynamic service selection [4], datacentre resource allocation [16] and dynamic power management [6].

Here we extend the applicability of RQV to distributed self-adaptive software. To this end, we introduce an RQV-driven approach for DEcentralised Control In Distributed sElf-adaptive software (DECIDE). DECIDE addresses two key objectives from the latest research roadmap for self-adaptive systems [21]:

1. *Decentralisation of control loops*, to eliminate the single point of failure created by centralised control loops, to improve the flexibility of self-adaptive systems, and to fulfil the original autonomic computing vision [18].
2. *Practical runtime verification and validation*, to guarantee compliance with the QoS requirements of mission-critical self-adaptive software.

To the best of our knowledge, DECIDE is the first approach that uses formal verification to simultaneously decentralise the control loop of self-adaptive systems, and provide guarantees on their compliance with QoS requirements.

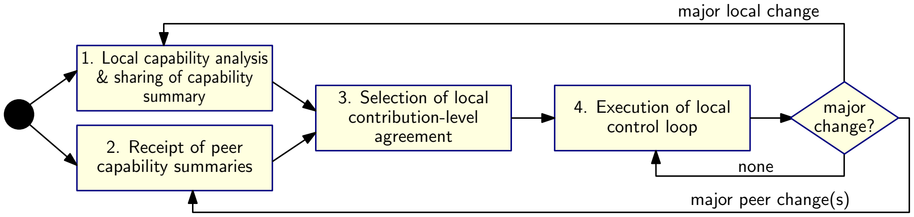

major local change
1. Local capability analysis & sharing of capability summary
2. Receipt of peer capability summaries
3. Selection of local contribution-level agreement
4. Execution of local control loop
major change?
none
major peer change(s)

Fig. 1: Decentralised self-adaptation workflow for a DECIDE component

Overview: Each component of a DECIDE system executes a decentralised control workflow comprising the four stages shown in Fig. 1. First, local RQV is used in a *local capability analysis* stage, to establish a *capability summary*, i.e., a finite set of alternative contributions the component could make towards achieving the system-level QoS requirements. This stage is executed infrequently (e.g., when the component joins the system), and the capability summary is shared with the peer components. The local computation or the receipt of a peer capability summary triggers the *selection of a local contribution-level agreement* (CLA). This CLA is one of the alternative contributions from the capability summary of the local component, chosen such that the system complies with its QoS requirements as long as each component achieves its CLA. Most of the time, the *execution of a local control loop* is the only DECIDE stage carried out by a component. Its purpose is to ensure compliance with the selected component CLA through RQV-driven local adaptation. Infrequently, events such as significant workload increases or failures of component parts render a DECIDE local control loop unable to achieve its CLA. These events are termed *major changes*, and require the computation and selection of new local CLAs.

Contributions: The original contributions of the paper are (a) a theoretical foundation for decentralising the control loops of distributed self-adaptive software; (b) an RQV method for devising component capability summaries in distributed self-adaptive systems; (c) a method for the decentralised selection of component-level agreements; and (d) a case study that used DECIDE to develop a simulated distributed embedded system in the unmanned marine vehicle domain.

Structure of the paper: Sections 2–3 introduce the theoretical background underlying DECIDE and the distributed embedded system used for its illustration and evaluation. The stages of DECIDE are presented in Section 4, and its implementation and evaluation are described in Section 5. Section 6 presents related work, and Section 7 summarises our findings and discusses future work directions.

2 PRELIMINARIES

DECIDE is applicable to systems that exhibit stochastic behaviour, and involves the runtime quantitative verification of formal models that describe the behaviour of their components. Here we present a class of such models called *continuous-time Markov chains* (CTMCs), and the temporal logic *continuous stochastic logic* (CSL), which is used to express the properties of CTMCs, as we will show for the running example we will introduce in Section 3. However, no change is required to use DECIDE with other types of probabilistic models, including discrete-time Markov chains [4, 5, 11] and probabilistic automata [16].

Definition 1. *A* continuous-time Markov chain *(CTMC) over a set of atomic propositions AP is a tuple*

$$M = (S, s_0, \mathbf{R}, L), \tag{1}$$

where:

- *S is a finite set of states, and $s_0 \in S$ is the initial state;*
- *$\mathbf{R} : S \times S \to [0, \infty)$ is a transition rate matrix such that for any state $s_i \in S$, the probability that the CTMC will transition from state s_i to another state within $t > 0$ time units is $1 - e^{-t \cdot \sum_{s_k \in S} \mathbf{R}(s_i, s_k)}$, and the probability that the new state is $s_j \in S$ is given by $\mathbf{R}(s_i, s_j) / \sum_{s_k \in S} \mathbf{R}(s_i, s_k)$.*
- *$L : S \to 2^{AP}$ is a labelling function.*

Quantitative or *probabilistic* model checkers operate on models expressed in a high-level, state-based language. Given a CTMC specified in this language, its representation (1) is derived automatically. We use the model checker PRISM [20], which supports the analysis of CTMCs extended with *costs/rewards*.

Definition 2. *A cost/reward structure over a CTMC with state set S is a pair of real-valued functions $(\underline{\rho}, \iota)$, where*

- *$\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is a* state reward function *that defines the rate $\underline{\rho}(s)$ at which the reward is obtained while the Markov chain is in state s;*
- *$\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a* transition reward function *that defines the reward obtained each time a transition occurs.*

Continuous stochastic logic (CSL) augmented with costs/rewards [19] is used to specify the quantitative properties to analyse for CTMC models.

Definition 3. *Let AP be a set of atomic propositions, $a \in AP$, $p \in [0, 1]$, I an interval in $\mathbb{R}$ and $\bowtie \in \{\geq, >, <, \leq\}$. Then a state formula Φ and a path formula Ψ in CSL are defined by the following grammar:*

$$\Phi ::= true | a | \Phi \wedge \Phi | \neg \Phi | P_{\bowtie p}[\Psi] | S_{\bowtie p}[\Psi]; \quad \Psi ::= X\Phi | \Phi \cup^{\leq I} \Phi \tag{2}$$

and the cost/reward augmented CSL state formulae are defined by the grammar:

$$\Phi ::= R_{\bowtie r}[I^{=T}] | R_{\bowtie r}[C^{\leq T}] | R_{\bowtie r}[F\Phi] | R[S] \tag{3}$$

CSL formulae are interpreted over states of a CTMC model. Path formulae only occur inside the probabilistic operator P and *steady-state* operator S, which define bounds on the probability of system evolution. For instance, a state s satisfies a formula $P_{\bowtie p}[\Phi]$ if the probability of the future evolution of the system meets the bound $\bowtie p$. For a path, the "next" formula $X\Phi$ holds if Φ is satisfied in the next state; the "bounded until" formula $\Phi_1 \cup^{\leq I} \Phi_2$ holds if before Φ_2 becomes true at time step $x \in I$, Φ_1 is satisfied continuously in the interval $[0, x) \cap I$. If $I = [0, \infty)$, the formula is termed "unbounded until". The notation $s \models \Phi$ and $M \models \Phi$ indicates that Φ is satisfied in state s and in the initial state of a CTMC model M, respectively. The semantics of the cost/reward operator R is as follows: $R_{\bowtie r}[I^{=T}]$ denotes the expected value of the reward at time instant T; $R_{\bowtie r}[C^{\leq T}]$ denotes the expected cumulative reward up to time T; $R_{\bowtie r}[F\Phi]$ gives the expected cumulative reward before reaching a state that satisfies Φ; and $R[S]$ is the average expected reward in the long-run.

3 RUNNING EXAMPLE

We will illustrate the application of DECIDE using a distributed multi-UUV (unmanned underwater vehicle) embedded system that extends our single-UUV system from [15]. UUVs are used for oceanic surveillance, survey and rescue operations, mine detection, and discovery of natural resources [24]. Enhancing UUV systems with self-adaptive capabilities is highly desirable to ensure their successful operation in the uncertain marine environment. This is particularly important given the limited ability to control UUV systems remotely, the criticality of their missions, and the need to minimise loss of expensive equipment.

We consider an n-UUV system deployed on a surveillance and data collection mission. The UUVs travel within proximity of each other, and the i-th UUV is equipped with $n_i > 0$ on-board sensors that can take periodic measurements of a characteristic of the ocean environment, e.g., dissolved oxygen, salinity or temperature. The l-th sensor of UUV i operates with varying rate $r_{il} \geq 0$, and the probability p_{il} that one of its measurements is sufficiently accurate for the purpose of the mission depends on the vehicle speed $sp_i \in [0, sp_i^{\max}]$. This is typical for such devices, e.g., the measurement error of sonars can be approximated by a normal distribution with zero mean and standard deviation that increases with speed. For each measurement taken, an amount of energy e_{il} is consumed. Each UUV can switch on and off its sensors individually to save battery power; these operations consume energy given by e_{il}^{on} and e_{il}^{off}, respectively. The UUV system must meet the following system-level QoS requirements:

R1: The n UUVs should take at least 1000 measurements of sufficient accuracy per 60 seconds of mission time.

R2: At least two UUVs should have switched-on sensors at any time.

R3: Subject to R1–R2 being satisfied, the system should minimise energy use (so that the mission can continue for longer).

In addition, each UUV i must satisfy the local QoS requirements below:

R4: The energy e_i used by sensors in a minute should not exceed $e_i^{\max}$ Joules.

R5: No sensor with accuracy probability below $p_i^{\min}$ should be used.

R6: Subject to R4–R5 being satisfied, the UUV should minimise the local cost function $w_1 e_i + w_2 sp_i^{-1}$, where $w_1, w_2 > 0$ are UUV-specific weights.

In a dynamic environment, each UUV i should adapt to changes in the operating rate of its sensors and to sensor failures, by continually adjusting:

a) the UUV speed sp_i;

b) the sensor configurations $x_{i1}, x_{i2}, \ldots, x_{in_i}$, where $x_{ij} = 1$ if the j-th sensor is switched on and $x_{ij} = 0$ otherwise,

so that the system- and the UUV-level QoS requirements are met at all times.

4 APPROACH

DECIDE distributed self-adaptive systems comprise $n > 1$ components. We use Cfg_i and Env_i to denote the set of possible configurations and the set of possible environment states for the i-th component, respectively. Thus, Cfg_i corresponds to parameters that the local control loop of component i can modify, and Env_i to parameters that the component can only observe. In addition, component i has $m_i \geq 1$ QoS attributes $\mathit{attr}_{i1} \in V_1$, $\mathit{attr}_{i2} \in V_2$, $\ldots$, $\mathit{attr}_{im_i} \in V_{m_i}$, where the value domain V_j of the j-th attribute could be $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$, etc. We further assume that the value of the j-th attribute depends on the current environment state $e \in \mathit{Env}_i$ and configuration $c \in \mathit{Cfg}_i$, and is given by:

$$\mathit{attr}_{ij}(e,c) = f_{ij}(e, c, M_i(e,c) \models \Phi_{ij}), \tag{4}$$

where M_i is a Markov model parameterised by the state of the environment the component operates in and the configuration selected by its local control loop, Φ_{ij} is a probabilistic temporal logic formula, and $f(\cdot,\cdot,\cdot)$ is a function that can be evaluated in O(1) time. The m_i attributes (Fig. 2) have the following roles:

1. Attributes $\mathit{attr}_{i1}, \mathit{attr}_{i2}, \ldots, \mathit{attr}_{im}$, $m < m_i$, are associated with the $m > 0$ QoS requirements of the DECIDE distributed system. Formally, the j-th system QoS requirement, $1 \leq j \leq m$, is specified as

   $$\mathit{expr}_j(\mathit{attr}_{1j}, \mathit{attr}_{2j}, \ldots, \mathit{attr}_{nj}) \bowtie_j \mathit{bound}_j \tag{5}$$

   where a non-exhaustive list of options for the expression expr_j, relational operator $\bowtie_j$ and bound bound_j is shown in Table 1.
2. Attribute $\mathit{attr}_{i,m+1}$ is a measure of the system-level cost associated with the current environment state and configuration of component i. Accordingly, $V_{m+1} = \mathbb{R}_+$ and the system-level cost $\sum_{i=1}^{n} \mathit{attr}_{i,m+1}$ needs to be minimised, subject to the m QoS requirements being satisfied.
3. Attributes $\mathit{attr}_{i,m+2}, \mathit{attr}_{i,m+3}, \ldots, \mathit{attr}_{i,m_i-1} \in \mathbb{B}$ represent component-level QoS requirements that are satisfied iff

   $$\mathit{attr}_{ij} = \mathit{true} \text{ for } j = m+2, m+3, \ldots, m_i - 1. \tag{6}$$
4. Attribute $\mathit{attr}_{i,m_i} \in \mathbb{R}_+$ is a measure of the component-level cost, which must be minimised subject to system and component QoS requirements being met.

Example 1. The set of configurations for UUV i of our distributed n-UUV system from Section 3 is $\mathit{Cfg}_i = \mathit{Sp}_i \times \{0,1\}^{n_i}$, where $(\mathit{sp}_i, x_{i1}, x_{i2}, \ldots, x_{in_i}) \in \mathit{Cfg}_i$ give the UUV speed sp_i and sensor configurations $x_{i1}, x_{i2}, \ldots, x_{in_i}$ selected by the local control loop. The set of environment states for UUV i is $\mathit{Env}_i = \mathbb{R}_+^{n_i}$, where $(r_{i1}, r_{i2}, \ldots, r_{in_i}) \in \mathit{Env}_i$ gives the measurement rates for the n_i sensors.

The Markov model $M_i(e,c)$ used to compute the QoS attributes of UUV i in (4) is obtained through the parallel composition of CTMC models of the n_i sensors: $M_i = M_{i1} \parallel M_{i2} \parallel \ldots \parallel M_{in_i}$. Fig. 3 depicts the model of the l-th sensor, when this sensor is switched on ($x_{il} = 1$) or off ($x_{il} = 0$), hence the transition from the initial state s_0 to state s_1 or s_6, respectively. The transition

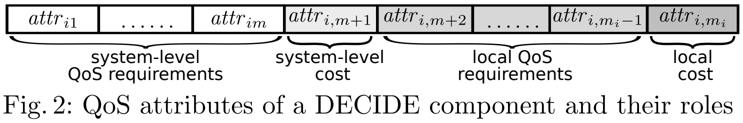


Fig. 2: QoS attributes of a DECIDE component and their roles

Table 1: Categories of DECIDE system-level QoS requirements from (5)

V_j	$expr_j(attr_{1j}, attr_{2j}, \ldots, attr_{nj})$	$\bowtie_j \in$	$bound_j \in$	Types of QoS requirements
$\mathbb{R}_+$	$\sum_{i=1}^{n} w_i attr_{ij}$, $w_i > 0$ weights	$\{<, \leq, \geq, >\}$	$\mathbb{R}_+$	throughput, energy usage, response time
$[0, 1]$	$\prod_{i=1}^{n} w_i attr_{ij}$, $w_i > 0$ weights	$\{<, \leq, \geq, >\}$	$[0, 1]$	reliability, availability
$\mathbb{B}$	$booleanExpr(attr_{1j}, \ldots, attr_{nj})$	$\{=, \neq\}$	$\mathbb{B}$	liveness, security

$s_1 \to s_2$ corresponds to a measurement being taken with rate r_{il}. With probability p_{il}, the measurement is accurate and M_i transitions to state s_3; otherwise it transitions to state s_4. This operation continues while the sensor is active, as modelled by the transition $s_5 \to s_1$. The CTMC is augmented with two reward structures, whose non-zero elements are shown in Fig. 3 in rectangular boxes, and dashed rectangular boxes, respectively. The first structure ("*energy*") associates the energy used to switch the sensor on (e_{il}^{on}) and off (e_{il}^{off}) and to perform a measurement (e_{il}) with the transitions that model these events. The second structure ("*measure*") associates a reward of 1 to accurate measurements.

Given the model M_i, the CSL formulae and the functions in Table 2 are used in (4) to establish the QoS attributes for requirements R1–R6. The $m = 2$ system-level requirements, R1 and R2, are given by the following instances of (5):

$$\text{R1:} \sum_{i=1}^{n} attr_{i1} \geq 1000 \qquad \text{and} \qquad \text{R2:} \bigvee_{1 \leq i_1 < i_2 \leq n} (attr_{i_1 2} \wedge attr_{i_2 2}) = true \qquad (7)$$

4.1 DECIDE Stage 1: Local capability analysis

During this DECIDE stage, each component uses runtime quantitative verification to assemble a summary of its capabilities, as formally defined below.

Definition 4. *A finite set $CS_i \subset V_1 \times V_2 \times \cdots \times V_{m+1}$ is an* α-confidence capability summary *for the i-th component of a DECIDE system iff, for any $(a_{i1}, a_{i2}, \ldots, a_{i,m+1}) \in CS_i$, the local control loop of the component can ensure that:*

(i) $attr_{ij} \bowtie_j a_{ij}$, for $1 \leq j \leq m$; (ii) $attr_{i,m+1} \leq a_{i,m+1}$; and

(iii) $attr_{ij} = true$, for $m + 1 < j \leq m_i - 1$

with probability at least $\alpha \in (0, 1)$.

The DECIDE method for calculating the α-confidence capability summary of system component i, $1 \leq i \leq n$, involves the local execution of the steps below.

1. *Configuration analysis:* Select $N_i > 0$ disjoint configuration subsets Cfg_i^1, Cfg_i^2, …, $Cfg_i^{N_i} \subset Cfg_i$ that correspond to different *modes of operation* for component i. What constitutes a mode of operation is component dependent. Examples include running different numbers of component instances, or operating with different degrees of accuracy. In our running example, UUV modes of operation correspond to different sensor sets being used.

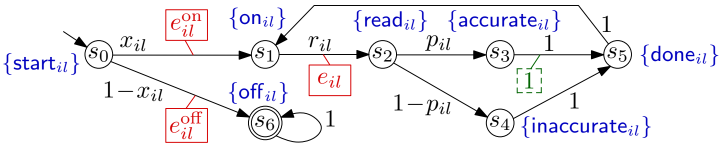


Fig. 3: CTMC model M_{il} of the l-th sensor from UUV i

2. *Environment analysis:* Identify subsets of environment states Env_i^1, Env_i^2, …, $Env_i^{N_i} \subseteq Env_i$ associated with the N_i configuration subsets, such that the probability that the environment state is in Env_i^k is at least α, for any $1 \leq k \leq N_i$. These subsets can be identical. However, in the general case, each configuration subset Cfg_i^k may render different areas of the environment state irrelevant, and DECIDE exploits this as illustrated in Example 2.
3. *Attribute analysis 1:* Check that for any $1 \leq k \leq N_i$ and for any $1 \leq j \leq m$ with $\bowtie_j \in \{=, \neq\}$, the QoS attribute $attr_{ij}(c,e)$ has a single value, a_{ij}^k, for all $(c,e) \in (Cfg_i^k, Env_i^k)$. When this is not the case, further partition the configuration set Cfg_i^k into disjoint subsets that satisfy this constraint. As shown in Table 1, one of the scenarios in which $\bowtie_j \in \{=, \neq\}$ is when $V_j = \mathbb{B}$. In this case, Cfg_i^k needs to be partitioned into two subsets. For other scenarios (e.g., when $V_j = \mathbb{R}_+$), DECIDE can be applied only if this operation partitions Cfg_i^k into a finite (and usually small) number of subsets. The rationale for this operation is that we want to associate each configuration set Cfg_i^k with a "bound" a_{ij}^k for each of $attr_{ij}$, $1 \leq j \leq m$, and the bounds a_{ij}^k are common values for QoS attributes $attr_{ij}$ for which $\bowtie_j \in \{=, \neq\}$.
4. *Attribute analysis 2:* For all $attr_{ij}$, $1 \leq j \leq m$, with $\bowtie_j \in \{<, \leq, \geq, >\}$, and for each configuration set Cfg_i^k, find simultaneous bounds $a_{ij}^k \in V_j$ such that
$$\forall e \in Env_i^k \bullet \exists c \in Cfg_i^k \bullet global(c,e) \wedge local(c,e), \tag{8}$$
where $global(c,e) = \bigwedge_{\substack{1 \leq j \leq m \\ \bowtie_j \notin \{=,\neq\}}} \left(attr_{ij}(c,e) \bowtie_j a_{ij}^k\right)$ and $local(c,e) = \bigwedge_{j=m+2}^{m_i - 1} attr_{ij}(c,e)$. When there is a single global QoS attribute $attr_{ij}$ with $\bowtie_j \in \{<, \leq, \geq, >\}$, its associated a_{ij}^k bound can be calculated as
$$a_{ij}^k = \begin{cases} \max\limits_{e \in Env_i^k} \ \min\limits_{c \in Cfg_i^k, local(c,e)} attr_{ij}(c,e), \text{ if } \bowtie_j \in \{<, \leq\} \\ \min\limits_{e \in Env_i^k} \ \max\limits_{c \in Cfg_i^k, local(c,e)} attr_{ij}(c,e), \text{ otherwise} \end{cases} \tag{9}$$
Otherwise, a multi-objective optimisation technique [10, 13] needs to be used.
5. *Cost analysis:* Calculate the cost upper bound
$$a_{i,m+1}^k = \max_{e \in Env_i^k} \ \min_{c \in Cfg_i^k, global(c,e) \wedge local(c,e)} attr_{i,m+1}(c,e).$$
6. *Capability summary assembly:* Use the a_{ij}^k bounds from steps 3–5 to assemble
$$CS_i = \{cs_i^1, cs_i^2, \ldots, cs_i^{N_i}\}, \tag{10}$$
where $cs_i^k = (a_{i1}^k, a_{i2}^k, \ldots, a_{i,m+1}^k)$, $1 \leq k \leq N_i$.

Table 2: QoS attributes for UUV i, where val_{ij} is the value of $M_i(e,c) \models \Phi_{ij}$

j	V_j	Φ_{ij}	$attr_{ij} = f_{ij}(e,c,val_{ij})$
1	$\mathbb{R}_+$	$R_{=?}^{\text{“}measure\text{”}}\left[C^{\leq 60}\right]$	val_{i1}
2	$\mathbb{B}$	$P_{\geq 1}\left[F\ \mathsf{on}_{i1} \vert \mathsf{on}_{i2} \vert \ldots \vert \mathsf{on}_{in_i}\right]$	val_{i2}
3	$\mathbb{R}_+$	$R_{=?}^{\text{“}energy\text{”}}\left[C^{\leq 60}\right]$	val_{i3}
4	$\mathbb{B}$	$R_{\leq e_i^{\max}}^{\text{“}energy\text{”}}\left[C^{\leq 60}\right]$	val_{i4}
5	$\mathbb{B}$	$\bigwedge_{l=1}^{n_i}\left(\mathsf{read}_{il} \Rightarrow P_{\geq p_i^{\min}}\left[X\ \mathsf{accurate}_{il}\right]\right)$	val_{i5}
6	$\mathbb{B}$	$R_{=?}^{\text{“}energy\text{”}}\left[C^{\leq 60}\right]$	$w_1 val_{i6} + w_2 sp^{-1}$

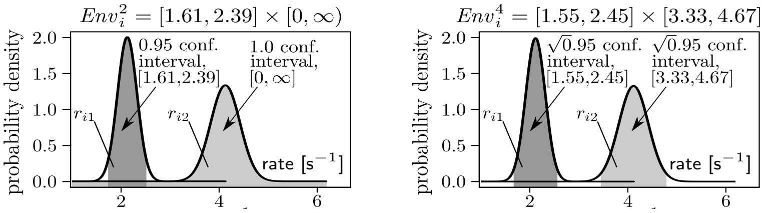


Fig. 4: Environment analysis for configuration sets Cfg_i^2 and Cfg_i^4 in Example 2

We are now ready for the following result, whose proof is available at `http://www-users.cs.york.ac.uk/~simos/DECIDE`.

Theorem 1. *The set CS_i in (10) is an α-confidence capability summary for component i of a DECIDE system.*

At the end of the local capability analysis stage of DECIDE, the local capability summary (10) is shared with the other components within the distributed system. For this purpose, we envisage DECIDE using recently emerged platforms for the engineering of distributed systems such as Kevoree [14] and DEECo [2].

Example 2. Suppose that the i-th UUV from our running example has $n_i = 2$ on-board sensors whose operating rates r_{i1} and r_{i2} are normally distributed with mean 2s^{-1} and standard deviation 0.2s^{-1}, and with mean 4s^{-1} and standard deviation 0.3s^{-1}, respectively. The UUV$_i$ environment state has the form (r_{i1}, r_{i2}), and the set of all environment states is $Env_i = [0, \infty]^2$. Also, assume that the UUV speed sp_i can be adjusted in the range $[1\text{m/s}, 5\text{m/s}]$. Hence, the UUV configuration set is $Cfg_i = [1, 5] \times \{0, 1\}^2$, where for any configuration $(sp_i, x_{i1}, x_{i2}) \in Cfg_i$, $x_{ij} = 1$ if sensor j is switched on and $x_{ij} = 0$ otherwise, for $j \in \{1, 2\}$. Finally, suppose that the bounds for local QoS requirements R4–R5 are $e_i^{\max} = 1000\text{J}$ and $p_i^{\min} = 0.9$, and that the energy used by the sensor operations are: $e_{i1} = 3J$, $e_{i1}^{\text{on}} = 15J$, $e_{i1}^{\text{off}} = 3J$, $e_{i2} = 2J$, $e_{i2}^{\text{on}} = 10J$, $e_{i2}^{\text{off}} = 2J$. An $(\alpha = 0.95)$-confidence capability summary for UUV i is built as follows:

1. *Configuration analysis*—A UUV mode of operation corresponds to using different subsets of sensors, so there are four configuration subsets: $Cfg_i^k = \{(sp_i, 0, 0) | sp_i \in [1, 5]\}$, $Cfg_i^2 = \{(sp_i, 1, 0) | sp_i \in [1, 5]\}$, $Cfg_i^3 = \{(sp_i, 0, 1) | sp_i \in [1, 5]\}$ and $Cfg_i^4 = \{(sp_i, 1, 1) | sp_i \in [1, 5]\}$.
2. *Environment analysis*—Assuming that the sensor rates r_{i1} and r_{i2} are independent, the environment state subsets Env_i^k, $1 \leq k \leq 4$, are obtained as Carthesian products of α_1 and α_2 confidence intervals for r_{i1} and r_{i2}, respectively, $\alpha_1\alpha_2 = 0.95$. When a single sensor $j \in \{1, 2\}$ is active for a configuration set Cfg_i^k, we use $\alpha_j = \alpha$ for it and $\alpha_{3-j} = 1$ for the inactive sensor when calculating Env_i^k. This allows the UUV to "promise" a stronger contribution towards the system requirements by disregarding the uncertainty in the state of switched-off sensors for configuration sets that have such sensors (Figure 4).
3. *Attribute analysis 1*—The relational operators for the $m = 2$ system requirements (7) are $\bowtie_1$='$\geq$' and $\bowtie_2$='$=$', so DECIDE checks that the second attribute from Table 2 takes a single value within each configuration set Cfg_i^k.

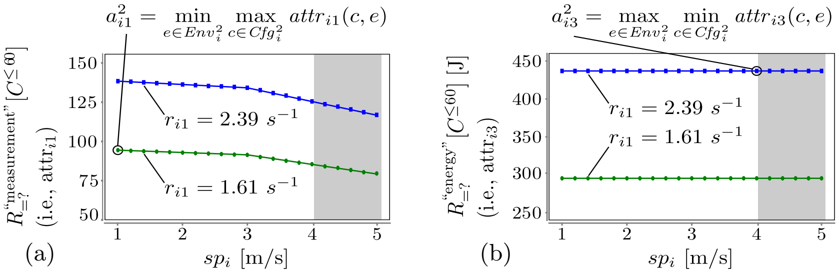


Fig. 5: Verification of Φ_{i1} and Φ_{i3} from Table 2; shaded areas correspond to configurations that violate local requirement R5

This check is successful because $attr_{i2} = false = a_{i2}^1$ for all configurations in Cfg_i^1 (as both sensors are switched off) and $attr_{i2} = true = a_{i2}^k$ for all configurations in Cfg_i^k, $2 \leq k \leq 4$. Hence, no partition of any Cfg_i^k set is required.

4. *Attribute analysis 2*—Requirement R1 in (7) is the only global requirement whose associated relational operator $\bowtie_1$ belongs to the set $\{<, \leq, \geq, >\}$. Thus, DECIDE uses RQV to derive the bounds a_{i1}^k in (9) for $1 \leq k \leq 4$. Fig. 5a shows the analysis that establishes a_{i1}^2 using the model checker PRISM [20]. The worst-case environmental scenario in Env_i^2 is when $r_{i1} = 1.61\text{s}^{-1}$; the highest number of measurements in this scenario (i.e., a_{i1}^2) is achieved for $sp_i = 1\text{m/s}$.
5. *Cost analysis*—As shown in Fig. 5b, the cost $attr_{i3}$ is constant for each Env_i^k, and the maximum cost, a_{i3}^k, corresponds to the highest sensor rate(s) in Env_i^k.
6. *Capability summary assembly*—The bounds a_{ij}^k, $1 \leq j \leq 3$, $1 \leq k \leq 4$, obtained in steps 3–5 are organised into the four-element capability summary $CS_i = \{(0, \mathit{false}, 5), (94, \mathit{true}, 435), (193, \mathit{true}, 535), (279, \mathit{true}, 984)\}$.

4.2 DECIDE Stage 2: Receipt of peer capability summaries

In this stage, the α-confidence capability summary (10) of a component is shared with all the other components within the system. DECIDE does not propose a new mechanism for sharing capability summaries. Instead, this DECIDE stage relies on the data sharing capabilities of recently emerged platforms for the engineering of distributed systems such as Kevoree [14] and DEECo [2].

4.3 DECIDE Stage 3: Selection of component contributions

In this stage, each of the n system components decides its contribution to the realisation of the system QoS requirements. To this end, each component uses the capability summaries $CS_1, CS_2, \ldots, CS_n$ to solve the optimisation problem:

$$\begin{array}{ll} \text{minimise} & \sum_{i=1}^{n} a_{i,m+1} \\ \text{subject to} & expr_j(a_{1j}, a_{2j}, \ldots, a_{nj}) \bowtie_j bound_j, 1 \leq j \leq m \\ \text{and} & (a_{i1}, a_{i2}, \ldots, a_{i,m+1}) \in CS_i, 1 \leq i \leq n \end{array} \tag{11}$$

Assuming the problem has a solution, the CLA for the i-th component is given by

$$cla_i = (a_{i1}, a_{i2}, \ldots, a_{i,m+1}) \tag{12}$$

from this solution, and we say that the i-th system component satisfies its CLA iff the QoS attributes of the component satisfy

$$\begin{array}{ll} attr_{ij} \bowtie_j a_{ij}, \text{ if } \bowtie_j \in \{<, \leq, \geq, >\} \\ attr_{ij} = a_{ij}, \quad \text{otherwise (i.e., if } \bowtie_j \in \{=, \neq\}) \end{array}, \text{ for all } 1 \leq j \leq m. \tag{13}$$

Remember from Section 4.1 that component configurations that satisfy (13) exist with probability at least α. We can now introduce the following theorem, whose proof is provided at `http://www-users.cs.york.ac.uk/~simos/DECIDE`.

Theorem 2. *Let cla_1, cla_2, ..., cla_n be the CLAs (12) of a DECIDE system with QoS requirements (5). If component i satisfies cla_i for all $1 \leq i \leq n$, then the system QoS requirements are satisfied.*

DECIDE does not prescribe how the optimisation problem (11) should be solved, as this is application specific. Depending on the nature of the DECIDE system and its requirements, the best way to obtain the component CLAs (12) may be by using an efficient dynamic programming or greedy algorithm, a metaheuristic or, when the solution space $CS_1 \times CS_2 \times \ldots \times CS_n$ is sufficiently small, by examining all options. The CLA calculation is performed independently by each system component (using the same deterministic method).

Example 3. Consider again our n-UUV system. The instance of the optimisation problem (11) solved by the DECIDE module running on each UUV is

$$\begin{array}{ll} \text{minimise} & \sum_{i=1}^{n} a_{i,3} \\ \text{subject to} & \sum_{i=1}^{n} a_{i1} \geq 1000 \text{ and } \bigvee_{1 \leq i_1 < i_2 \leq n} (a_{i_1 2} \wedge a_{i_2 2}) = \mathit{true} \\ \text{and} & (a_{i1}, a_{i2}, a_{i3}) \in CS_i, 1 \leq i \leq n \end{array} \qquad (14)$$

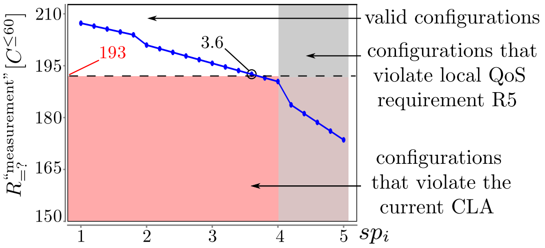
Our implementation (Section 5) casts this as a multiple-choice knapsack problem [17] and solves it using an efficient $\mathrm{O}(n^2)$ dynamic programming algorithm. We provide this algorithm at `http://www-users.cs.york.ac.uk/~simos/DECIDE`.

4.4 DECIDE Stage 4: Execution of local control loop

Most of the time, this is the only DECIDE stage being executed. The local control loop ensures that each component meets its CLA and local requirements by implementing the RQV-driven approach to developing (single control loop) self-adaptative software that we co-introduced in [6, 9], summarised in [3], and extended and used successfully in multiple application domains [4, 5, 15, 16].

The local control loop of component i uses RQV to establish the value of $M_i(e, c) \models \Phi_{ij}$ in (4), $1 \leq j \leq m_i$, either periodically and/or after events associated with environment or component changes. The aim is to verify if the QoS attributes of the component continue to satisfy the component CLA (12) and local requirements (6), and, if this is not the case, to identify a new configuration that does. The search for such new configurations starts with the configuration subset Cfg_i^k associated with the component CLA. When no configuration in Cfg_i^k is suitable, the search is extended to the entire configuration space Cfg_i. A component that is no longer able to meet its CLA and local requirements is affected by a "major change", and its capability summary is recalculated (Section 4.1). Describing single-control-loop RQV-driven adaptation is outside the scope of this paper, and we refer the reader to [3] for an overview of the approach.

Example 4. Suppose the CLA selected for the two-sensor UUV$_i$ from Example 2 is $(193, \mathit{true}, 535)$. The local control loop will adjust the UUV configuration in response to changes in the sensor rates r_{i1}, r_{i2} such that the UUV achieves at least 193 accurate measurements, has at least an active sensor, and consumes at



$$
\begin{aligned}
&\mathit{attr}_{i1} = \mathit{val}_{i1} = 194 \text{ (depicted left)}\\
&\mathit{attr}_{i2} = \mathit{val}_{i2} = \mathit{true}\\
&\mathit{attr}_{i3} = \mathit{val}_{i3} = 434(\mathrm{J})\\
&\mathit{attr}_{i4} = \mathit{val}_{i4} = \mathit{true}\\
&\mathit{attr}_{i5} = \mathit{val}_{i5} = \mathit{true} \text{ iff } sp_i \leq 4 \left(\tfrac{\mathrm{m}}{\mathrm{s}}\right)\\
&\mathit{attr}_{i6} = w_1 \mathit{val}_{i6} + w_2 sp_i^{-1} =\\
&\qquad = w_1 \cdot 434 + w_2 sp_i^{-1}
\end{aligned}
$$

Fig. 6: RQV of Φ_{i1}– Φ_{i6} from Table 2

most 535J for each 60s of operation. Fig. 6 shows the RQV results when $r_{i2} = 3.68\mathrm{s}^{-1}$, for the configuration subset $\mathit{Cfg}_i^3 = \{(sp_i, 0, 1) | sp_i \in [1, 5]\}$ associated with this CLA; as sensor 1 is switched off, the value of r_{i1} is irrelevant. Given these results, the control loop selects the configuration $(sp_i, x_1, x_2) = (3.6, 0, 1)$, which meets the CLA and local requirements, and minimises the local cost attr_{i6}.

4.5 Major changes

Major changes trigger the execution of other DECIDE stages than the local control loop, as shown in Figure 1. There are two types of major changes.

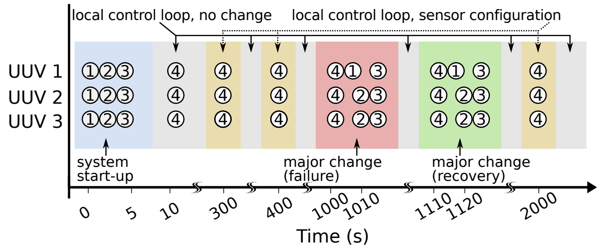
A *local major change* occurs within a component when: (a) the local control loop cannot find a configuration that satisfies its CLA or local requirements; (b) failures within the component make certain modes of operation unavailable; and (c) the capability summary becomes overly conservative, due to a more favourable environment than anticipated or to recovery from a previous failure. In these scenarios, the component re-executes the local capability analysis stage.

A *peer major change* occurs when another component (a) joins the system; (b) undergoes a local major change; or (c) leaves the system. Component i learns about peer major changes of types (a) and (b) when it receives a new capability summary. For the last type of change, DECIDE requires that component failures are notified by the communication and synchronisation platform underpinning the interactions between components. This capability is supported by platforms such as Kevoree [14] and DEECo [2], and can be readily exploited by DECIDE.

5 EVALUATION

Implementation — To evaluate DECIDE, we implemented a fully-fledged simulator for the multi-UUV self-adaptive system from our running example. We built our simulator on top of the open-source MOOS-IvP middleware (`http://oceanai.mit.edu/moos-ivp`), a widely used C++ platform for the implementation of autonomous applications on unmanned marine vehicles [1].

The simulator integrates the standard MOOS-IvP publish-subscribe and visualisation components with our MOOS component that implements the four DECIDE stages. Over 90% of our component is new code that implements the DECIDE local capability analysis, receipt of peer capability summaries, CLA selection and major change identification within local control loops. The rest is reused from our previous work on a single-UUV self-adaptive system with a centralised control loop [15]. The code for our multi-UUV simulator, the experimental results summarised in this section, and a video recording of a typical simulation are available at `http://www-users.cs.york.ac.uk/~simos/DECIDE`.



DECIDE stage	CPU use [ms]	bytes sent	bytes recv.
1	38.3	71*	0
2	0	0	71*
3	0.7	0	0
4	25.6	0	0

* per peer UUV

Fig. 7: Execution of DECIDE stages 1–4 during major changes and local sensor changes, and mean CPU and communication overheads for a three-UUV mission

Experimental setup — To evaluate DECIDE, we carried out a broad range of experiments using our multi-UUV system simulator. The system characteristics varied in these experiments include the number of UUVs n and UUV sensors n_i, $1 \leq i \leq n$, and the confidence level α used to assemble UUV capability summaries. To examine the impact of different types of failure, the experiments were seeded with failure patterns including of failures of sensors, sudden significant reductions in sensor measurement rates (i.e., sensors not meeting their specification) and failures of entire UUVs. All experiments were carried out on a 2.6GhZ Intel Core i5 Macbook Pro computer with 16GB memory, running Mac OSX 10.9.

Typical simulation scenario — Fig. 7 shows the execution of the DECIDE stages during key moments of a simulated 2000s mission carried out by a three-UUV system. Each UUV had three sensors, and the requirements enforced by DECIDE were requirements R1–R6 from Section 3. As shown in Fig. 7, the CPU overheads for the RQV-based local capability analysis and control loop (DECIDE stages 1, 4) and the knapsack problem solving in CLA selection (stage 3) are all negligible at under 40ms each, or below 0.4% when the local control loop is executed every 10s. The communication overhead, 71 bytes per peer UUV per major change, is very low too, even for a typical inter-UUV bandwidth of 0.5–5Kbps [23].

Adaptation effectiveness — In all experiments, the system recovered after sensor failures and performance drops, and UUV failures within 800ms from the moment when the last periodically run local control loop of an UUV started executing, for a typical inter-UUV bandwidth of 2.5Kbps. Thus, if the local control loop runs every 5s, the time to recovery was below 5.8s.

Adaptation efficiency — To assess the efficiency of the DECIDE self-adaptation decisions, we compared the number of measurements taken and the energy consumed by the three-UUV DECIDE system with the values of the same metrics for an "ideal" system (Table 3). In this "ideal" system (a) the sensor rates never varied from their nominal values; (b) the globally optimal set of sensors satisfying requirements R1–R6 were used at all times; and (c) all UUVs travelled with the minimum speed of 1m/s, to maximise the fraction of measurements that were accurate. This "ideal" system cannot be implemented in practice, but has the useful property that any practical system will use more measurements and more energy than it does. Accordingly, the results in Table 3 show that DECIDE successfully decentralised the control loop of the UUV system with a modest loss in efficiency compared to any other solution that might be possible.

Role of confidence level α — Higher confidence levels make the component capability summaries (10) more conservative. This increases the system-level cost (e.g., the energy use for our system, see Table 3), but reduces the number of false positives from local control loop checks for major changes, as a fraction of α of the checks find the component operating in an expected environment state.

Scalability — As shown in Fig. 8, the two RQV-based DECIDE stages (i.e., the local capability analysis and the local control loop) use the same small amount of CPU time irrespective of the size of the n-UUV system. The $\mathrm{O}(n^2)$ CPU time taken by the CLA selection stage stays below 200ms for systems of up to 32 UUVs. In contrast, using a centralised control loop that applies RQV to the entire system model $M_1 \parallel M_2 \parallel \ldots \parallel M_n$ takes over 4200s for $n = 2$ and is unfeasible for $n > 3$. The CPU time shown in Fig. 8 for the RQV of a complete model of a three-UUV system (i.e., 983.5 days) is an estimate we obtained based on the average verification time over a small subset of representative configurations from the configuration set that would need to be verified by this control loop.

Threats to validity — We identified several threats to external validity. First, as we evaluated DECIDE in a single case study, our approach may not be applicable to other systems because it may not be possible to cast their QoS attributes and requirements into the pattern given by (4)–(6) in Section 4. To limit this threat, we distilled this pattern from the growing body of research on RQV-driven self-adaptation in service-based, cloud-deployed and embedded software systems [3–6, 9, 11, 15, 16]. Second, for other systems it may not be possible to identify α-confidence subsets of environment states. This threat is mitigated by the fact that DECIDE can operate with approximations of such subsets, which impact only the frequency of major changes. Finally, major changes may occur too frequently, leading to unacceptable overheads and "jitter" in component reconfigurations. DECIDE can alleviate this by increasing the α confidence level (i.e., being more conservative), but our approach is not intended for systems with a high churn rate. Threats to internal validity originate from how experiments were performed. To reduce them, we developed our simulator using the well-established UUV software platform MOOS-IvP, we examined a wide range of scenarios, and we repeated all experiments many times.

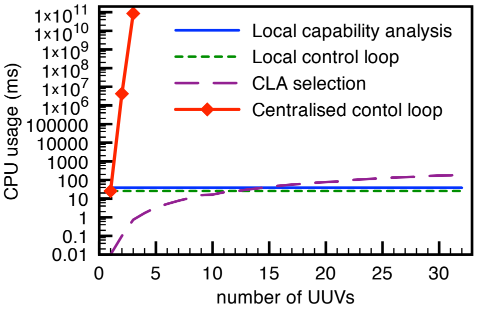


Fig. 8: Scalability analysis

Table 3: Comparison of DECIDE with the "ideal" system (average results over 10 experiments)

confidence level α	additional energy use	additional measurements
0.90	+18.26%	+12.54%
0.95	+18.30%	+12.58%
0.99	+20.62%	+9.97%

6 RELATED WORK

To the best of our knowledge, DECIDE is the first approach to using runtime quantitative verification (RQV) to decentralise the control loops of self-adaptive systems. Although RQV has attracted a lot of attention since its recent introduction in [3, 6, 9], the research so far (e.g., [4, 5, 11, 15, 16]) has focused on its use in centralised control loops. This is feasible only for systems whose stochastic models are small enough to be analysed fast and with acceptable overheads. Also, for distributed systems such as those in service-based applications [4, 5], centralised control introduces a single point of failure. DECIDE addresses both limitations. As all RQV steps analyse component models, our approach extends the applicability of RQV to larger component-based systems. Also, the use of RQV in DECIDE does not introduce a single point of failure, since the control loop of a component continues to operate when a peer component fails.

Decentralised-control self-adaptive systems have been developed using many other approaches, e.g. as multi-agent [7, 12, 25] and service-based systems [22]. Due to space constraints, we mention here only a few of these approaches, selected as representative for the field. Most of them take adaptation decisions using optimisation heuristics that cannot provide the strong guarantees required in mission-critical applications, e.g. bio-inspired [7], market-based [22] and gossip-style [25] heuristics. In contrast, DECIDE guarantees that the decentralised-control system meets its QoS requirements in the presence of changes, recovering from failures whenever feasible. More recent research explored the use of formal methods to guarantee that decentralised-control self-adaptive systems meet their functional requirements [8, 12, 26]. Our work complements this research, since DECIDE focuses on the QoS requirements of distributed self-adaptive systems.

7 CONCLUSION

We presented an approach to using runtime quantitative verification (RQV) to develop self-adaptive distributed systems with decentralised control loops. RQV-based decentralised control ensures that distributed systems developed using our approach continue to meet their QoS requirements after failures and environment changes. Compared to the current use of a centralised RQV control loop for the same purpose [3, 4, 6, 9, 11, 15, 16], our new approach achieves this: (a) with overheads that are several orders of magnitude lower; (b) scalably to much larger system sizes; (c) without introducing a single point of failure; and (d) with only a modest increase in system-level costs (18–21% in our case study).

In future work, we will assess the effectiveness of DECIDE in other domains, and examine its scalability for systems with larger component models (e.g., UUV systems with more sensors per UUV). In addition, we are extending DECIDE with support for using interface models as component QoS attributes, with assume-guarantee RQV used to verify system QoS properties as in [16].

Acknowledgments

This paper presents research sponsored by the UK MOD. The information contained in it should not be interpreted as representing the views of the UK MOD, nor should it be assumed that it reflects any current or future UK MOD policy.

References

1. M. Benjamin et al. Autonomy for unmanned marine vehicles with MOOS-IvP. In *Marine Robot Autonomy*, pages 47–90, 2013.
2. T. Bures et al. Deeco: An ensemble-based component system. In CBSE 2013.
3. R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Comm. ACM* 55(9):69–77, 2012.
4. R. Calinescu, L. Grunske, M. Kwiatkowska, et al. Dynamic QoS management and optimization in service-based systems. *IEEE Trans. Softw. Eng.* 37:387–409, 2011.
5. R. Calinescu, K. Johnson, and Y. Rafiq. Developing self-verifying service-based systems. In ASE'13, pages 734–737, 2013.
6. R. Calinescu and M. Z. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In ICSE'09, pages 100–110, 2009.
7. G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos Self-Organization in Multi-Agent Systems In *The Knowledge Eng. Rev.*, 20(2):165-189, June 2005.
8. N. D'Ippolito et al. Hope for the best, prepare for the worst: Multi-tier control for adaptive systems. In ICSE'14, pages 688–699, 2014.
9. I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by runtime parameter adaptation. In ICSE'09, pages 111–121, 2009.
10. K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. In TACAS'07, pages 50–65, 2007.
11. A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In ICSE'11, pages 341–350, 2011.
12. M. Fisher, L. Dennis, and M. Webster. Verifying autonomous systems. *Comm. ACM* 56(9):84–93, 2013.
13. V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In TACAS'11, pages 112–127, 2011.
14. F. Fouquet et al. Kevoree modeling framework (KMF): Efficient modelling techniques for runtime use. *CoRR*, abs/1405.6817, 2014.
15. S. Gerasimou, R. Calinescu, A. Banks. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. In SEAMS 2014.
16. K. Johnson, R. Calinescu, and S. Kikuchi. An incremental verification framework for component-based software systems. In CBSE '13, pages 33–42, 2013.
17. H. Kellerer, U. Pferschy, and D. Pisinger. The multiple-choice knapsack problem. In *Knapsack Problems*, pages 317–347, 2004.
18. J. Kephart, D. Chess. The vision of autonomic computing. *Computer* 36(1), 2003.
19. M. Kwiatkowska. Quantitative verification: models, techniques and tools. In ESEC-FSE companion '07, pages 449–458, 2007.
20. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: verification of probabilistic real-time systems. In CAV'11, pages 585–591, 2011.
21. R. Lemos et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32, 2013.
22. V. Nallur and R. Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Trans. Softw. Eng.* 39(5):591–612, 2013.
23. S. Redfield. Cooperation between underwater vehicles. In M. L. Seto, editor, *Marine Robot Autonomy*, pages 257–286, 2013.
24. M. Seto, L. Paull, and S. Saeedi. Introduction to autonomy for marine robots. In *Marine Robot Autonomy*, pages 1–46, 2013.
25. D. Sykes, J. Magee, and J. Kramer. Flashmob: Distributed adaptive self-assembly. In SEAMS '11, pages 100–109, 2011.
26. D. Weyns et al. FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems. *ACM Trans. Aut. Adapt. Syst.* 7(1), 2012.