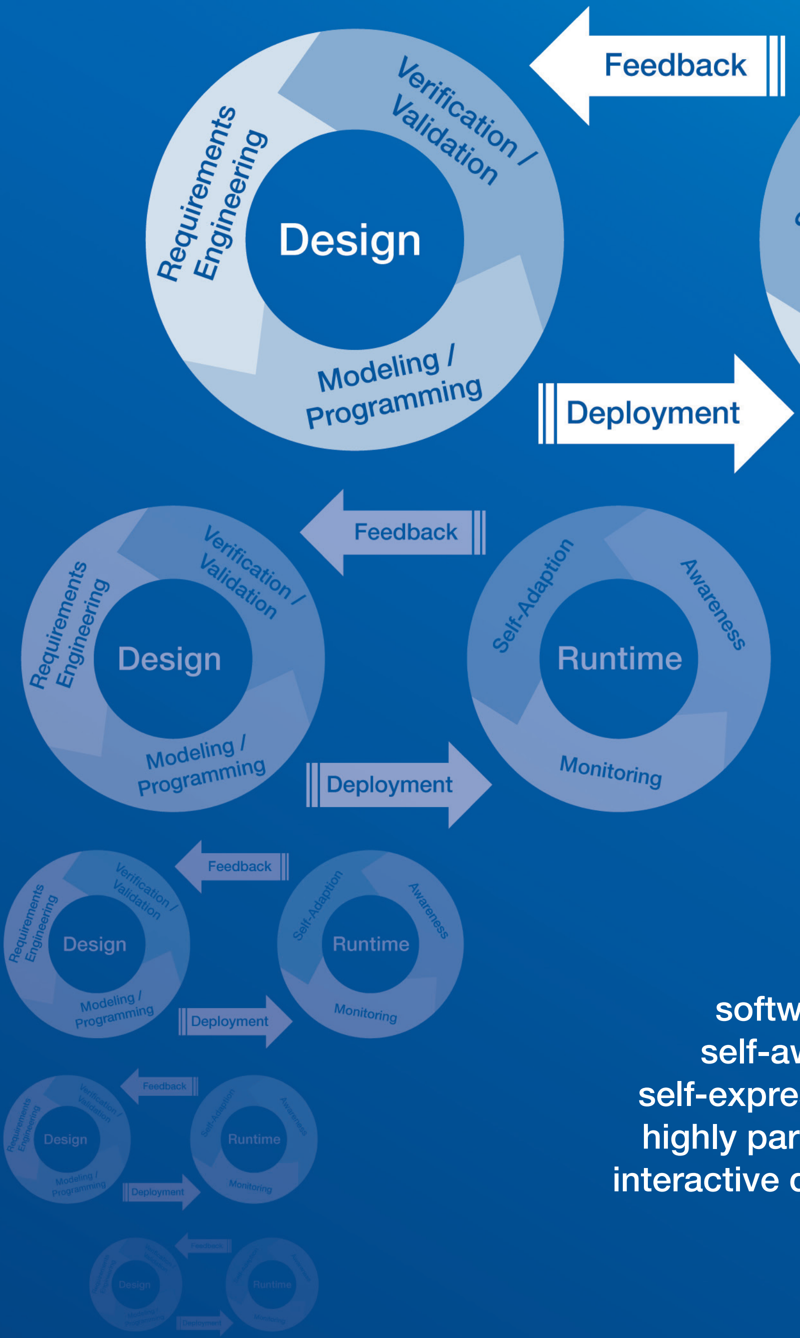
ascens

autonomic service-component ensembles



software engineering for
self-aware, self-adaptive,
self-expressive, open-ended,
highly parallel, collective and
interactive distributed systems

ascens meeting – Limerick, Ireland – July 2012

ascens
autonomic service-component ensembles

Martin Wirsing
Coordinator
wirsing@pst.ifi.lmu.de

Ludwig-Maximilians-Universität München

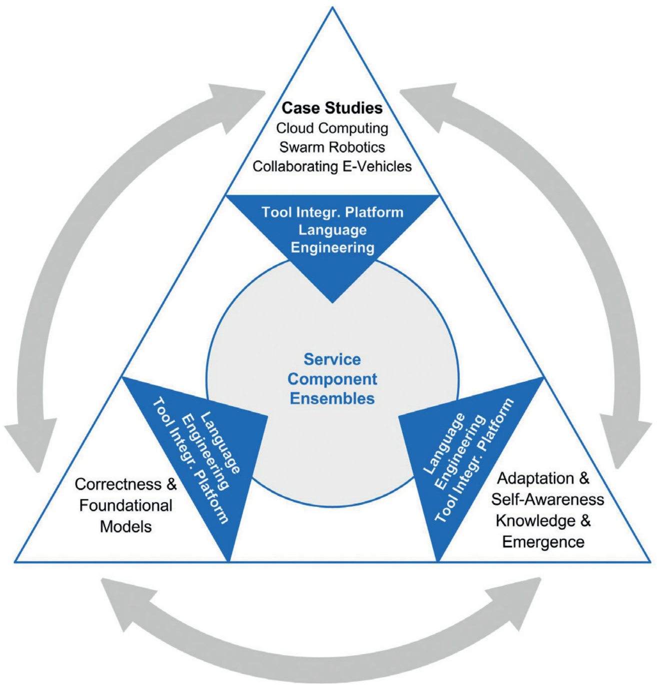
www.ascens-ist.eu

At runtime, the more autonomy a system exhibits, the less obvious that autonomic behavior appears to outside observers. Thus to be sure about the correct functioning of such systems it is necessary to support their development through appropriate methods and tools which can guarantee not only that an autonomic system really does what it is supposed to do, but also that important constraints of the environment are never violated.

The IST-FET Integrated Project **ascens** has developed a novel comprehensive approach to engineer autonomic self-adaptive systems – so-called ensembles – which is both, pragmatic and formal.

Pragmatic orientation means building autonomic systems that do practical things, like autonomic robot swarms performing rescue operations, autonomic cloud computing platforms transforming numerous small computers into a supercomputing environment, or autonomic e-mobility supporting ensembles of cooperative e-vehicles.

ascens offers a range of foundational theories and methods that support modeling, formal reasoning, validation and verification of complex controlled systems, monitoring and dynamic adaptation of autonomic systems, both at design and at runtime. Furthermore, the ensemble development life cycle includes a third feedback loop that enables design changes based on the system‘s and environment‘s awareness obtained during runtime.



Requirements Engineering for Self-Adaptive Systems

ARE - Autonomy Requirements Engineering

Autonomic self-adaptive systems extend regular software-intensive systems upstream with special self-managing objectives (or self-* objectives) that provide autonomy features in the form of a system‘s ability to automatically discover, diagnose, and cope with various problems. This ability depends on a system’s degree of autonomicity, quality and quantity of knowledge, awareness and monitoring capabilities, and quality characteristics such as adaptability, dynamicity, robustness, resilience, and mobility.

ARE starts with the creation of a goals model that represents system objectives and their inter-relationships. In the next phase, we work on each one of the goals along with elicited environmental constraints to come up with self-* objectives providing autonomy requirements for achieving these goals. The autonomy requirements are derived in the form of goal-supportive and alternative self-* objectives, along with required capabilities and quality characteristics. Finally, we specify the autonomy requirements with KnowLang, a framework dedicated to knowledge representation for self-adaptive systems.

Lero - University of Limerick

ESA ESTEC, the European Space Research and Technology Centre

Emil Vassev
emil.vassev@lero.ie

www.ascens-ist.eu/are

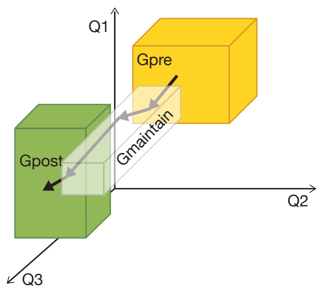
SOTA - State of the Affairs

SOTA is a method for specifying requirements by tracing the evolution of an ensemble’s “state of affairs” over time. In SOTA, the state space of an ensemble contains all parameters “Q_i” that may affect its behavior or capabilities. At each instance, the ensemble occupies a single point in this state space; the ensemble‘s activity over time is therefore expressed by a trajectory through the state space.

Requirements are specified as goals which correspond to regions of the state space that the ensemble’s trajectory must eventually reach (achieve goals “G_{post}”), regions in which it must stay throughout the execution (maintain goals “$G_{maintain}$”) or regions that it must avoid (avoid goals). A goal becomes active when its precondition “G_{pre}” is encountered; this may happen only once or repeatedly during the ensemble‘s lifetime.

Università degli Studi di Modena e Reggio Emilia

Franco Zambonelli
franco.zambonelli@unimore.it
www.ascens-ist.eu/sota



GEM - General Ensemble Model

Ludwig-Maximilians-Universität München

Matthias Hölzl
hoelzl@pst.ifi.lmu.de

www.ascens-ist.eu/gem

GEM is a mathematical formalization of the SOTA approach that gives precise semantics to SOTA models and provides means of specifying model properties in various logics, such as the higher-order logic of the Prototype Verification System (PVS) or temporal logics.

GEM enables developers to analyze requirements models using mathematical techniques. For example, a SOTA/GEM model may be used to derive an adaptation strategy for a swarm of robots operating in an adversarial environment by applying concepts from evolutionary game theory.

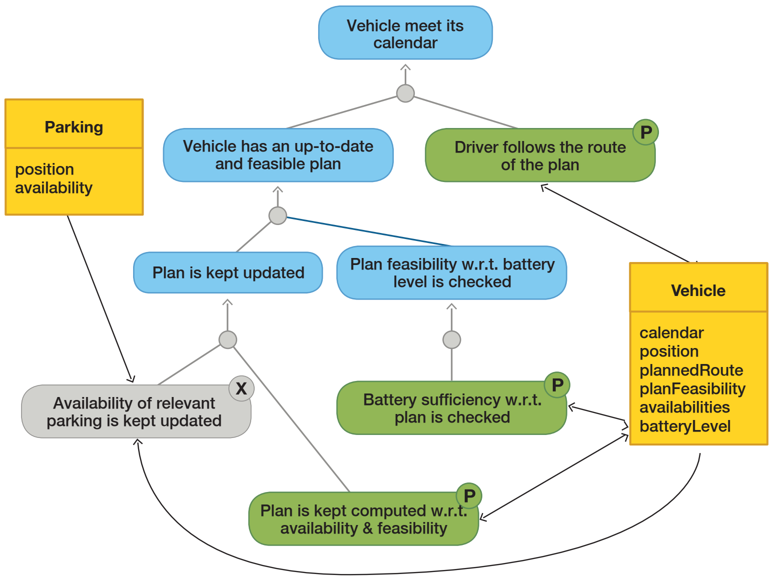
IRM - Invariant Refinement Method

Charles University of Prague

Tomáš Bureš
bures@d3s.mff.cuni.cz

www.ascens-ist.eu/irm

IRM is a requirements-oriented design method that facilitates modeling of ensemble-based systems. The main idea of IRM is to capture the high-level goals and requirements in terms of invariants (graphically represented as rounded rectangles), which describe the desired state of the system-to-be at every time instance. Invariants are to be maintained by the coordination of the different system components (graphically represented as rectangles).


Vehicle meet its calendar
Parking
position
availability
Vehicle has an up-to-date and feasible plan
Driver follows the route of the plan
P
Plan is kept updated
Plan feasibility w.r.t. battery level is checked
Vehicle
calendar
position
plannedRoute
planFeasibility
availabilities
batteryLevel
Availability of relevant parking is kept updated
X
Battery sufficiency w.r.t. plan is checked
P
Plan is kept computed w.r.t. availability & feasibility
P

As a design decision, top-level invariants are iteratively decomposed into more concrete sub-invariants, forming a decomposition graph with traceability of design decisions.

The decomposition process ends when the leaf invariants represent a detailed design of the system implementation – either in terms of local component behavior, corresponding to a component process (denoted by the “P” decoration of the invariant), or in terms of component interaction, corresponding to an ensemble (denoted by an “X” decoration).

Languages for Autonomic Systems

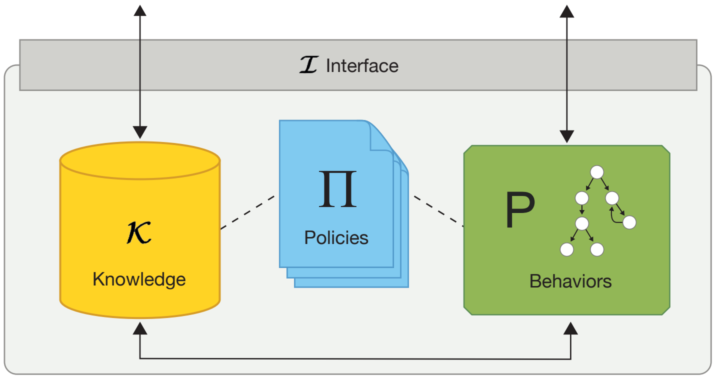
SCEL - Service Component Ensemble Language

SCEL is a formal language that takes a holistic approach to programming autonomic computing systems. In SCEL, systems are structured as ensembles of autonomic components.

The language provides programmers with a complete set of linguistic abstractions for programming the behavior of components and the formation of ensembles, and for controlling the interaction among them. These abstractions permit autonomic systems to be described in terms of behaviors, knowledge and aggregations, by complying with specific policies, and to support programming context-awareness, self-awareness and adaptation.

Each SCEL component is equipped with an interface, consisting of a collection of attributes describing its features (e.g., identity, functionalities, spatial coordinates, group memberships, trust level, response time, etc.).

Attributes are used by components to dynamically organize themselves into ensembles. Specifically, components can single out communication partners by using predicates over their attributes, thus permitting a sort of attribute-based communication. Therefore, SCEL ensembles are not rigid fixed networks, but rather highly flexible structures where components' linkages are dynamically established.



The solid semantic grounding of SCEL lays the basis for developing logics, tools and methodologies for formally reasoning about system behavior in order to establish qualitative and quantitative properties of both the individual components and their ensembles.

IMT Advanced Studies Lucca

Università degli Studi di Firenze

Rocco De Nicola
rocco.denicola@imtlucca.it

www.ascens-ist.eu/scel

jRESP - Runtime Environment for SCEL Programs

Università degli Studi di Firenze

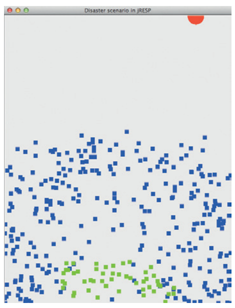
Michele Loreti
michele.loreti@unifi.it

www.ascens-ist.eu/jresp

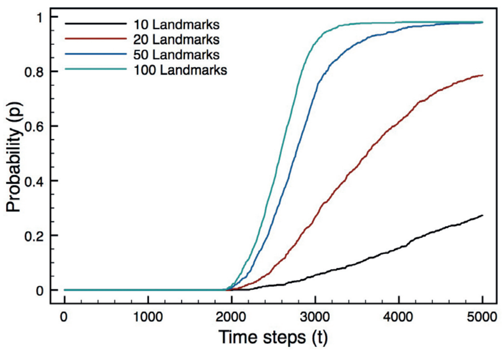
The jRESP framework provides an API that permits SCEL's linguistic constructs for controlling the computation and interaction of autonomic components to be used in Java programs, and for defining the architecture of autonomic and adaptive systems and ensembles.

jRESP also provides specific components that can be used to simulate SCEL programs. The screenshot shows a simulation run of a robotics scenario: A set of landmarks (blue squares in the picture) randomly walk to find a victim (depicted as a red circle). When the victim is found all the landmarks dynamically create an ensemble that allow the workers (green squares in the picture) to reach the victim and rescue it.

The jRESP simulation environment integrates a statistical model-checker that can be used to estimate the probability of reaching specific goals. The result of such a model checking analysis is shown below: The probability to save the victim within t time units is studied when the number of landmarks varies from 10 to 100.



Disaster scenario



StocS - Stochastic SCEL

ISTI CNR

IMT Advanced Studies Lucca

Università degli Studi di Firenze

Diego Latella
diego.latella@isti.cnr.it

www.ascens-ist.eu/stocs

StocS extends SCEL by modeling the durations of action executions as random variables with negative exponential distributions, thus obtaining continuous-time Markov chains as the underlying semantics model. The behavioral semantics underlying StocS is the so-called Network-Oriented one, where the distributed nature of action execution is addressed explicitly.

Our stochastic variants adopt the same language syntax of SCEL, or restrictions thereof, but denote different underlying stochastic models with different levels of granularity.

Other stochastic variants of SCEL are addressed in the context of the Quanticol EU Project. Such extensions are important for the analysis of the performance aspects of ensemble-based systems. Providing suitable Markovian semantics for predicate-based ensemble languages poses a number of design challenges regarding the temporal ordering of multicast and information request actions that differ considerably from traditional process algebras.

FACPL - Formal Access Control Policy Language

The FACPL language is the basis of a user-friendly, feasible and effective approach for developing, operating and maintaining policy-based autonomic systems. The language permits the expression of high-level policies regulating various aspects of a computer system, e.g., access control, resource usage and adaptation. It has a compact and intuitive syntax and is endowed with a denotational semantics providing a full formal account of the security model.

FACPL specifications are hierarchically structured in terms of FACPL elements, i.e., rules, policies and policy sets. These elements specify a name, the effect of a positive evaluation (i.e., permit or deny), target and conditions for applicability, the algorithm for combining the results of the evaluation of the contained elements, and a set of obligations, i.e., supplemental actions as for example updating a log file, sending a message, setting an attribute.

IMT Advanced Studies Lucca

Università degli Studi di Firenze

Rosario Pugliese
rosario.pugliese@unifi.it

www.ascens-ist.eu/facpl

```
{ pep: deny-biased
  pdp: permit-overrides
    PolicySet Create_Policies { permit-overrides
         target:
         equal ( "CREATE" , action / action-id )
         policies:
         Policy SLA_Type1 < deny-unless-permit
             target:
             (equal ( "P_1" , subjekt / profile-id ) || equal ( "P_2" ,
             subjekt / profile-id )) && equal ( "TYPE_1" , resource / vm-type )
             rules:
             Rule hyper_1 ( permit target:
                 less-than-or-equal ( 1 , system / hyper1.availableResources )
                 obl:
                 [ permit M create ( "HYPER_1" , system / vm-id , "TYPE_1" ) ]
             )
             Rule hyper_2 ( permit target:
                 less-than-or-equal ( 1 , system / hyper2.availableResources )
                 obl:
                 [ permit M create ( "HYPER_2" , system / vm-id , "TYPE_1" ) ]
             )
             obl:
             [ deny O warning ( "Not enough available resource for TYPE_1 VMs"
             ) ]
         >
    }
}
```

The evaluation of a request with respect to a FACPL element triggers the processing of the element. If the element‘s target does not match the request, the element does not apply.

Otherwise, in case of policies and policy sets, the processing proceeds by recursively evaluating the enclosed elements and by composing their resulting decisions through the specified combining algorithm; in the case of rules, the processing proceeds by evaluating the condition, if present, and by returning the rule’s effect as a decision. The final decision is then established on the basis of the result of obligation discharge.

FACPL Supporting Tools

IMT Advanced Studies Lucca

Università degli Studi di Firenze

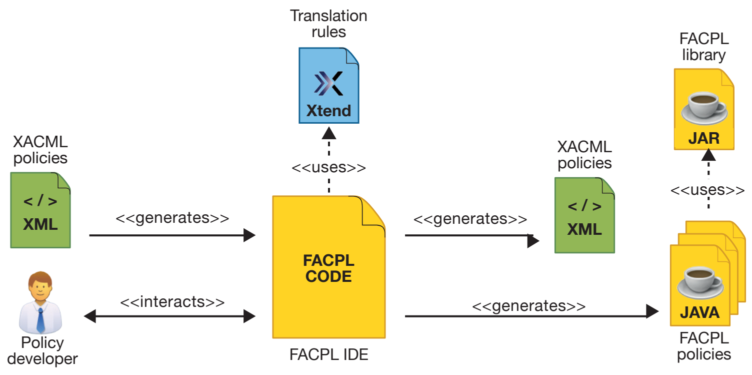
Andrea Margheri
andrea.margheri@unifi.it

www.ascens-ist.eu/facpl

FACPL is equipped with a powerful Integrated Development Environment (IDE) and a Java library, supporting policy developers and system administrators in the tasks of specifying, validating and enforcing policies specified in FACPL.

Policy developers can use the IDE, in the form of an Eclipse plugin, for specifying the desired policies in FACPL syntax, by taking advantage of the supporting features provided by the environment. Then, according to the rules defining the language's semantics, the IDE automatically produces a set of Java classes implementing the FACPL policies.

The Java FACPL library provides the compile- and run-time support for validating and enforcing the generated Java policies in real systems. Furthermore, the toolchain offers full interoperability with the well-established XACML standard for access control systems.



FACPL Integration in SCEL

IMT Advanced Studies Lucca

Università degli Studi di Firenze

Francesco Tiezzi
francesco.tiezzi@imtlucca.it

www.ascens-ist.eu/pl4scel

The SCEL language is designed according to the principle of separation of concerns, thus decoupling functional aspects from management ones. The latter aspects are regulated by policies, which provide a refinement process of components behavior to guarantee the accomplishment of specific tasks or satisfaction of specific properties. To enhance flexibility and better support self-management in different application domains, SCEL is parametric with respect to the policy language.

The FACPL policy language has then been integrated into SCEL to provide a complete language for programming and policing autonomic systems. In the resulting language, it is possible, for example, to define policies implementing adaptation strategies by exploiting specific actions that are produced at runtime as an effect of policy evaluation. These actions are executed as part of components' behaviors to enforce system adaptation.

Furthermore, policies can depend on the values of components' attributes (reflecting the status of components and their environment) and can be dynamically replaced to better react to system changes.

Modeling Self-Awareness and Adaptation

Reconfigurable and Dynamic Connectors

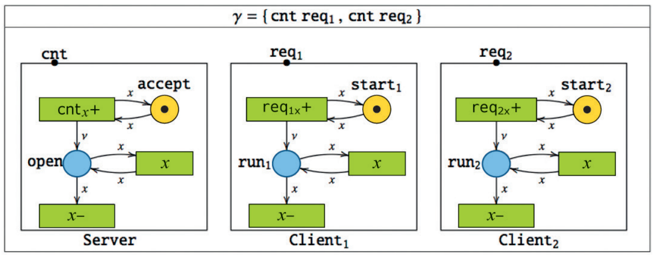
Università di Pisa

Universidad de Buenos Aires

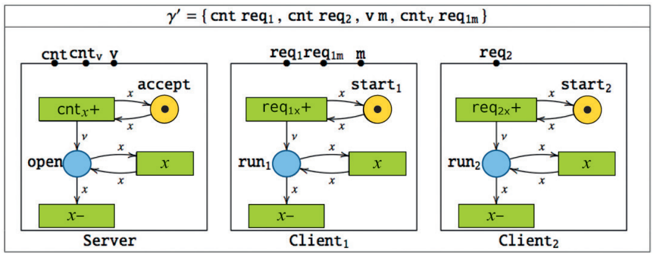
Roberto Bruni
bruni@di.unipi.it

www.ascens-ist.eu/rdc

Component-based design relies on the separation of concerns between coordination and computation: components are equipped with a set of ports and connectors that impose suitable constraints on the communications over the ports. The evolution of an ensemble can be seen as if played in rounds: at each round, the components try to interact through their ports and the connectors allow/disallow some of the interactions, informing the components about the decision.



(a) Inicial State


$\gamma' = \{\,\mathtt{cnt}\ \mathtt{req}_1\,,\ \mathtt{cnt}\ \mathtt{req}_2\,,\ \mathtt{v}\ \mathtt{m}\,,\ \mathtt{cnt}_v\ \mathtt{req}_{1m}\,\}$
cnt cntᵥ v
req₁ req₁ₘ m
req₂
accept
start₁
start₂
cntₓ+
req₁ₓ+
req₂ₓ+
open
run₁
run₂
x
x−
Server
Client₁
Client₂

(b) First Synchronisation

Due to the high dynamism of autonomic component ensembles, connectors need to be empowered with mechanisms for resource- and network-awareness, as well as adaptation, reflection and reconfigurability.

The theoretical foundations and expressive power of several classes of connectors with different degrees of dynamism have been investigated, focusing on BIP systems and Petri nets with boundaries. In particular, novel flavors of reconfigurable and dynamic connectors have been defined, where it is possible to model and analyze systems whose set of allowed interactions can change at runtime, and where the creation/ elimination of ports and interactions is also supported.

For example, a dynamic BIP system with one server and two clients (see top figure) evolves to a configuration in which one of the client and the server have established some dedicated ports for interacting (see bottom figure).

White-box Adaptation

Università di Pisa

IMT Advanced Studies Lucca

Andrea Corradini
andrea@di.unipi.it

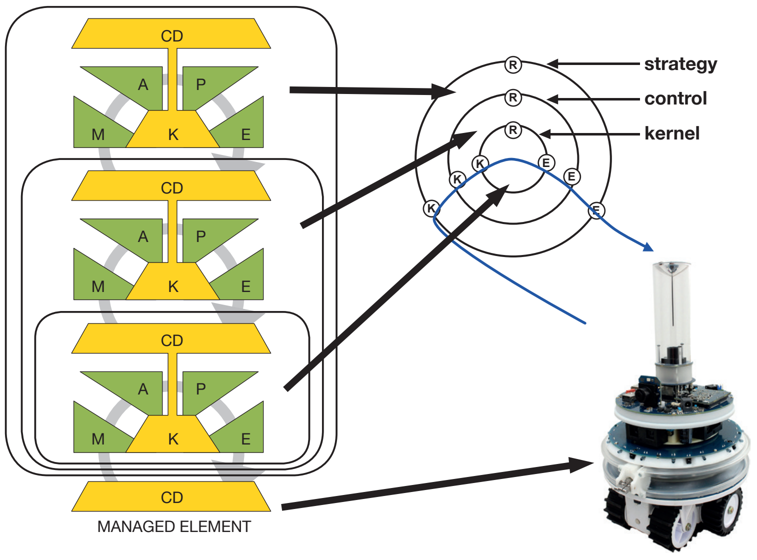
www.ascens-ist.eu/maia

A software system operating in unpredictable environments must be adaptive, modifying its own behavior in response to changes in its operating environment. Unfortunately, there is no agreed foundational model for adaptation. This is due to the inherent difficulty of subsuming both the external manifestations of adaptive systems (black-box adaptation) and the internal mechanisms that realize adaptation (white-box adaptation) in a coherent view.

In our conceptual, white-box-based, approach which requires the identification of so-called "control data" (CD) within the system, adaptation is defined as the runtime modification of such data. This provides an unambiguous definition of adaptation, allowing us at the same time to see the same system with different adaptation capabilities, depending on the chosen perspective.

This idea has been formalized with a variant of a classical game model for open systems, Interface Automata, yielding Adaptable Interface Automata (AIA). The key feature of such automata is control propositions, a subset of the atomic propositions labeling the states, imposing a clear separation between the ordinary, functional behavior and the adaptive one.

Control propositions are exploited in the analysis of adaptive systems, focusing on various notions like adaptability, control loops, and control synthesis. An implementation of AIAs in Maude, called MAIA, allows one to specify AIAs, to draw them, and to perform operations such as product, composition, decomposition and control synthesis.



The picture shows an "adaptation tower" where each control component performs a Monitor-Analyze-Plan-Execute loop using shared Knowledge (a MAPE-K loop), triggering adaptations of the lower managed component through the modification of its control data.

MESSI - Design and Performance Evaluation of Self-Assembly Strategies

Università di Pisa

IMT Advanced Studies Lucca

Andrea Vandin
a.vandin@soton.ac.uk

www.ascens-ist.eu/messi

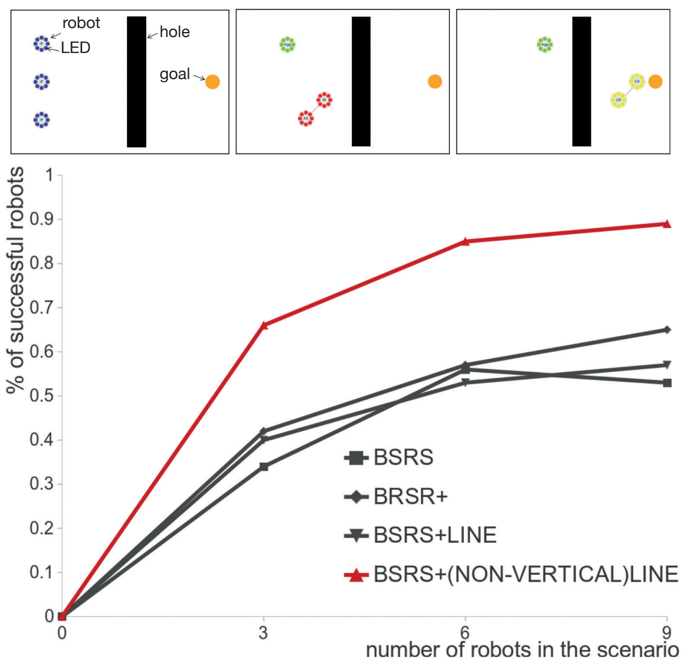
The methodology instantiates the white-box approach for adaptation, and has been developed to specify and analyze adaptive systems in Rewriting Logic. Validation has been performed for self-assembly strategies, a mechanism allowing groups of simple entities to act as a single complex entity exhibiting emergent behaviors. Notable examples include bacteria or insect swarms, modular and self-assembling robots, and software components with dynamic coupling mechanisms.

The distinguishing features of this approach are:

- adaptation based on the notion of control data;
- a hierarchical architecture to modularize the design;
- computational reflection as the main adaptation mechanism;
- probabilistic rule-based specifications and quantitative verification techniques to specify and analyze the adaptation logic.

The approach has been implemented in the Maude-based tool MESSI, which allows us to easily model robotic self-assembly strategies exploiting a predefined library of basic robotic controllers, debug and validate them via animated simulations, and estimate their quantitative properties via the statistical model checker MultiVeStA.

The picture concerns a scenario where robots self-assemble to cross a hole; it shows three states of a simulation (top), and the expected number of robots successfully crossing the hole with a varying of the number of robots and of the assembly strategy.



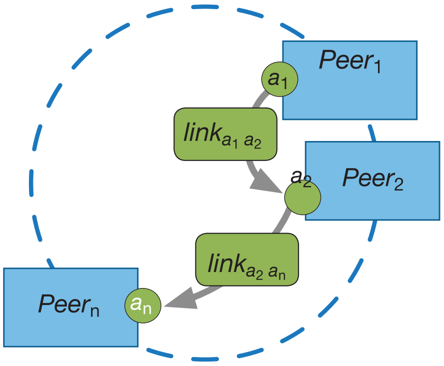
NCPi - Network-Aware Process Calculi

Università di Pisa

Matteo Sammartino
sammarti@di.unipi.it

www.ascens-ist.eu/ncpi

NCPi is a proper extension of the pi-calculus with an explicit notion of a network: network links and nodes are represented as names, in full analogy with ordinary pi-calculus names, and observations are routing paths through which data is transported. It provides a convenient framework for modeling systems with programmable network infrastructure, such as Peer-to-Peer (P2P) overlay networks.



NCPi comes in two versions, with different purposes: a basic one and a concurrent one.

The basic version is closer to the pi-calculus, and it has been used to investigate operational models with explicit network resource allocation. Two classes of models have been considered: coalgebras over presheaves and History-Dependent (HD-) automata. In particular, we have given a well-behaved categorical model of network resource allocation in a coalgebraic setting. Exploiting a standard categorical equivalence, we have derived a History-Dependent (HD-)Automaton from the coalgebra. HD-automata are nominal automata that, in many cases, admit a minimal representative with finitely-many states, thus suitable for verification. The mentioned categorical equivalence is very general and in principle could be used for other resource-aware calculi.

The concurrent version has been introduced to describe more realistic routing behavior. It has several additional features with respect to the basic version, the most important one being the possibility of observing multiple concurrent transmissions, which makes the semantics compositional.

This concurrent version of NCPi has been used to model the P2P architecture Pastry, which is part of Autonomic Cloud. In particular, some components and operations of peers have been modelled: routing data structures and their operations, reconfiguration due to joining peers and the provision of routing functionalities to applications. Moreover, a simple Distributed Hash Table has been modelled, where lookups are represented as routing paths from the peer that has invoked the lookup to the one responsible for the target key. A convergence property of routing has been proved both at the peer level, to prove correctness of the join procedure, and at the DHT level, to show that lookups always reach their destinations.

Energy-Aware Ensembles for Regenerative Power Production

The introduction of electric power production from renewable sources requires a proactive role of prosumers (producers/consumers) - a kind of ensemble. An efficient controller has been defined for delaying/anticipating the production/consumption of a prosumer following the real-time, distributed power market model proposed by the German project DEZENT.

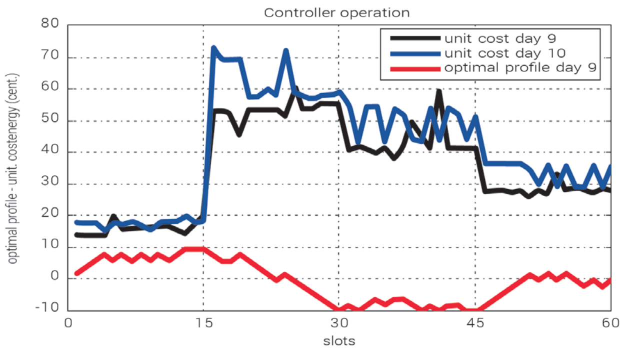
Università di Pisa

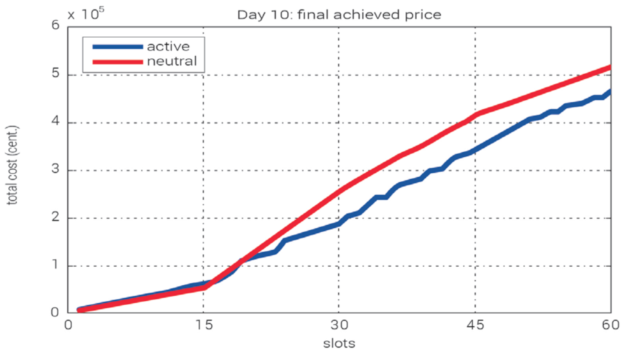
Ugo Montanari
ugo@di.unipi.it

www.ascens-ist.eu/eae

The market model determines how steep the cost curve should be using a reinforcement learning technique. The controller capitalizes on the (possibly very) different costs of the energy at different times of the day by concentrating energy consumption/production at the cheapest/most expensive times. The controller chooses the best energy profile among those acceptable in terms of existing energy storage items. They typically include batteries, but also biofuel consumption and heating/cooling delays.

Extended experimental studies have been carried out based on the project simulator and on the Java implementation of the optimal controller. Parameters include the type of experiment (active or neutral, i.e., placebo-like for comparison) and the scenario, i.e., if a request or offer is predominant in the market.





In the first graphic, the two upper curves display the cost of energy at different times on two consecutive days. They show the possible variations due to the market model. The lowest curve shows the optimal profile based on the cost of the first day. In the second graphic, it is shown that, in spite of the difference in cost between the two days, the profile computed in terms of the cost of the first day, when employed on the second day, is remarkably cheaper that the neutral profile.

Verification Techniques for Self-Aware Systems

UJF-Verimag

Saddek Bensalem
saddek.bensalem@imag.fr

Jacques Combaz
jacques.combaz@imag.fr

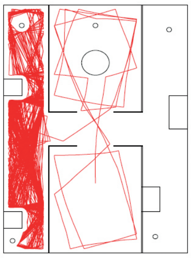
www.ascens-ist.eu/smcbip

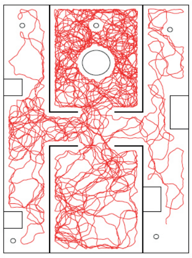
SMC-BIP - Statistical Model-Checking

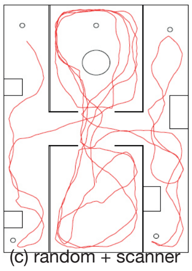
Swarm robotics systems usually operate under uncertain conditions (unknown environment, failures, etc.) so that their behavior can only be estimated quantitatively in terms of performance metrics (e.g., average failure rate of the robots, expected energy consumption).

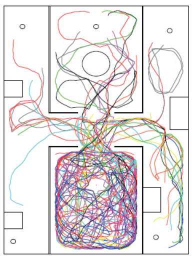
Statistical-model checking (SMC) is a "verification-inspired" approach that performs quantitative analysis of a system. Like model-checking, it uses formally-defined models from which it explores the reachable states. In contrast to standard model-checking, SMC does not require exhaustive computation of the reachable states to conclude about a given property. It answers quantitative questions based on partial state-space coverage, and evaluates confidence in such results based on stochastic models. SMC tools usually stop the analysis when the desired degree of confidence (provided as an input parameter) is reached.

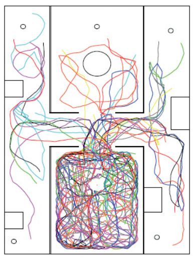
The statistical model-checking tool SMC-BIP has been applied to a swarm robotics scenario in which marXbot robots are deployed in a given arena to find victims. The stochastic model built for such a scenario includes a faithful model of the sensing capabilities of the marXbot. By using SMC-BIP, the performance of different algorithms used for implementing individual robot behavior can be compared which has proven to be very helpful for optimizing solutions to the scenario.


(a) straight


(b) random


(c) random + scanner


(a) landmarking


(b) landmarking + communication

Compositional Verification

UJF-Verimag

Saddek Bensalem
saddek.bensalem@imag.fr

www.ascens-ist.eu/compver

Formal verification corresponds to establishing a proof that a given system satisfies properties characterizing its correctness. It considers a (formal) model describing all possible system behaviors, and a means of proving correctness for all of them. To cope with the inherent complexity of such a task, it is always beneficial to perform verification at design time on high-level models, and to generate correct-by-construction implementations from those models.

Compositional verification is also a means to address state-space explosion and complexity in general. It relies on the characterization of local properties of the system, which avoids monolithic verification of the composed system. Such properties are then combined efficiently to establish global properties. In addition, it can be applied incrementally; i.e., if components are added to a system, the verification process re-uses properties already established for existing components.

For timed systems, verification is even more difficult since components' behavior and their correctness depend also on time. The major problem when verifying timed systems compositionally is capturing the relations between the local timing of the components induced by components' synchronizations.

Compositional verification has been successfully applied to a non-trivial coordination protocol for cooperating robots. It has been shown that the proposed protocol is safe for a subset of values of its parameters.

GMC - Gimple Model Checker

Charles University of Prague

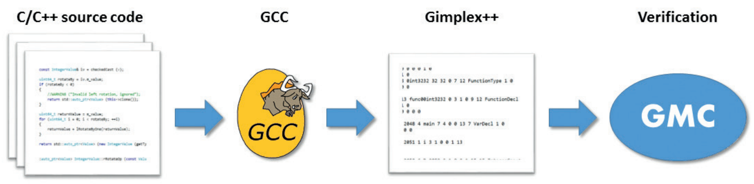
Jan Kofron
jan.kofron@d3s.mff.cuni.cz

www.ascens-ist.eu/gmc

GMC is an explicit model checker for the C and C++ languages. It is built upon the Gimplex++ representation of source code, an extension of the intermediate representation being used in the GCC project; the intermediate representation is called Gimple — that is where the GMC name stems from.

GMC is able to check the sources in C and the C++ language given that they do not contain calls to unsupported library functions. It supports verification of multi-threaded programs.

Even though support for POSIX threads is implemented, GMC provides general support for threading libraries, restricting the support incorporated to providing mapping of the thread functions (create thread, thread join, wait, ...) to the real function names in the form of a text file. GMC can verify that in any thread interleaving, no deadlock appears and no assertion stated in the code is violated.



MultiVeStA - Statistical Model Checking for Discrete Event Simulators

IMT Advanced Studies Lucca

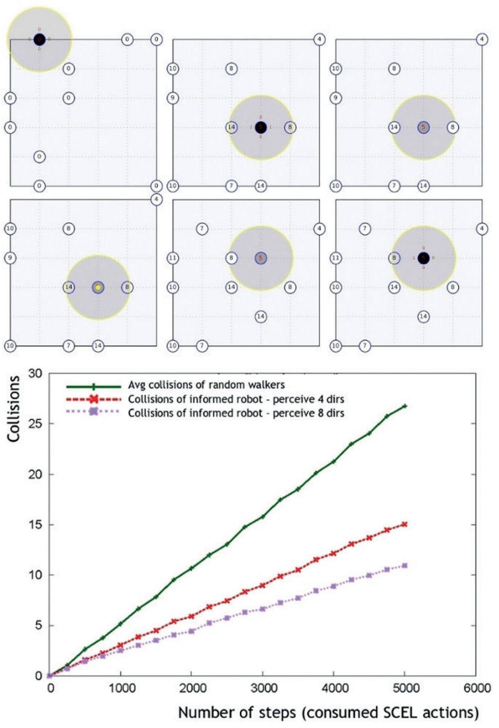
University of Southampton

Andrea Vandin
a.vandin@soton.ac.uk

www.ascens-ist.eu/multivesta

MultiVeStA is an efficient statistical analysis tool which can be easily integrated with existing discrete event simulators, enriching them with distributed Statistical Model Checking capabilities. MultiVeStA offers: a clean way to integrate discrete event simulators; a language (MultiQuaTEx) to compactly express many systems properties at once; the efficient estimation of the expected values of a MultiQuaTEx expression with respect to an user-specified confidence interval; the plot of the results in a minimal GUI; a client-server architecture to distribute simulations.

MultiVeStA extends the VeStA and PVeStA tools, and has been used to analyze SCEL specifications of collision-avoidance robotic scenarios; self-assembling robotic scenarios; volunteer cloud scenarios; reputation-based cloud scenarios; crowd-steering scenarios; public transportation systems.



The picture refers to the collision-avoidance robotic scenario, consisting of a group of robots moving in an arena. The top part of the picture depicts six intermediate states with nine random walker robots (the white circles), and an informed one (the black circle), which perceives the position of robots in its surrounding environment and selects the best direction to minimize collisions with other robots.

Two kind of informed robots are considered, with smaller and wider perception ranges. As expected, the plot in the bottom part of the picture shows that random walkers (green plot) perform more collisions than informed robots (red and pink plots for smaller and wider perception range), and a wider perception range allows for a reduction in collisions.

Knowledge Representation and Reasoning

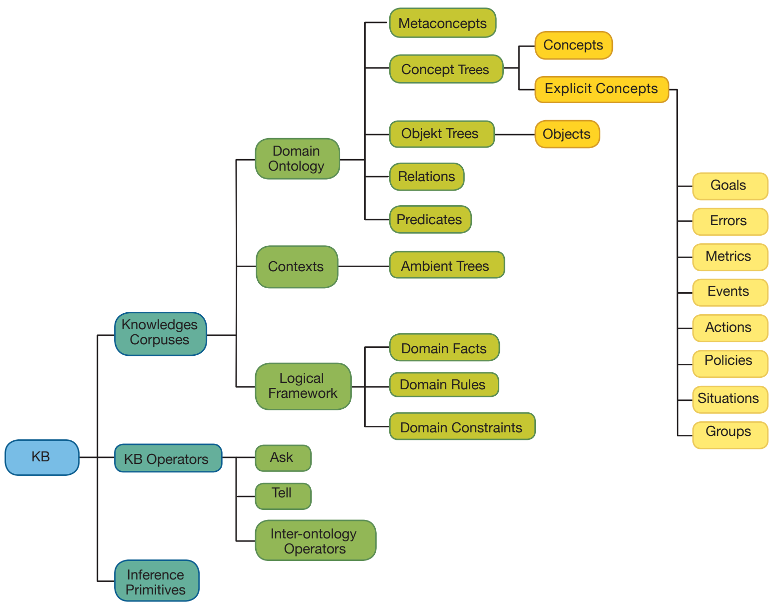
KnowLang - Knowledge Representation Language

Lero - University of Limerick

Emil Vassev
emil.vassev@lero.ie

www.ascens-ist.eu/knowlang

KnowLang is a formal specification language providing a comprehensive specification model that addresses the problem of knowledge representation for self-adaptive systems. The complexity of the problem necessitates the use of a specification model where knowledge can be presented at different levels of abstraction and grouped by following both hierarchical and functional patterns. The language imposes a multi-tier specification model where a knowledge base (KB) composed of layers of ontologies, operators, and inference primitives is specified.



KnowLang specifies self-* objectives through special policies associated with goals, situations, actions, metrics, etc. The self-* objectives define knowledge about what the system should do when particular situations arise when pursuing a system goal.

KnowLang policies are specified as individual concepts providing behavior (often concurrent). A policy has a goal, policy situations, policy-situation relations, and policy conditions mapped to policy actions where the evaluation of the conditions may eventually (with some degree of probability) imply the realization of actions. Policy situations may trigger (or imply) a policy in compliance with the policy-situation relations. The self-adaptive behavior requires relations to be specified to connect policies with situations over an optional probability distribution where a policy might be related to multiple situations and vice versa.

KnowLang Reasoner

Lero - University of Limerick

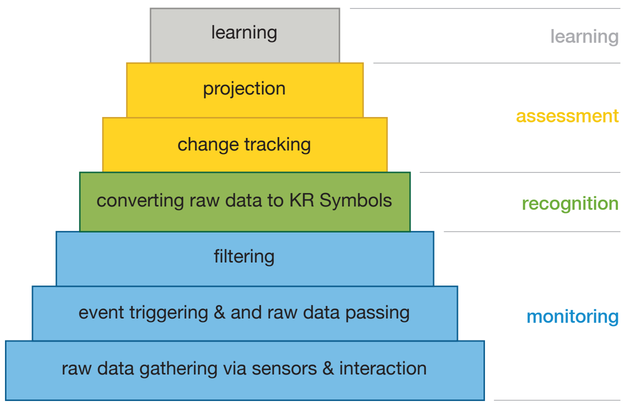
Emil Vassev
emil.vassev@lero.ie

www.ascens-ist.eu/knowlangr

The KnowLang Reasoner supports reasoning about self-adaptive behavior and provides a KR (knowledge representation) gateway via a set of special ASK and TELL Operators.

The reasoner operates in the KR Context, a context formed by the represented knowledge and outlined by the KB (knowledge base). The TELL Operators feed the KR Context with important information driven by errors, executed actions, new sensory data, etc. This helps the reasoner to update the KB with recent changes in both the system and the execution environment.

The system uses ASK Operators to receive recommended behavior where knowledge is used against the perception of the world in order to generate appropriate actions in compliance to some goals and beliefs. In addition, ASK Operators may provide the system with awareness-based conclusions about the current state of the system or the environment, and ideally with behavior models for self-adaptation.


learning
projection
change tracking
converting raw data to KR Symbols
filtering
event triggering & and raw data passing
raw data gathering via sensors & interaction
learning
assessment
recognition
monitoring

As part of the KnowLang Reasoner, we have developed a structured mechanism implementing awareness by taking into consideration different stages of the awareness process. This mechanism is built over a complex chain of functions pipelining the stages of the awareness process such as: raw data gathering; data passing; filtering; conversion; assessment; projection; and learning.

For awareness-based conclusions, the KnowLang Reasoner implements a Pyramid of Awareness mechanism forming the mechanism that converts raw data (facts, measures, raw events, etc.) into conclusions, problem prediction and eventually may trigger learning.

The different pyramid levels represent awareness functions that can be grouped into four function groups monitoring, recognition, assessment, and learning, all structured in a special awareness control loop. Aggregation can be included as a subtask at any function level. The learning functionality is implemented as probability redistribution in both policies and situations.

Iliad & Poem - Learning & Reasoning at Runtime

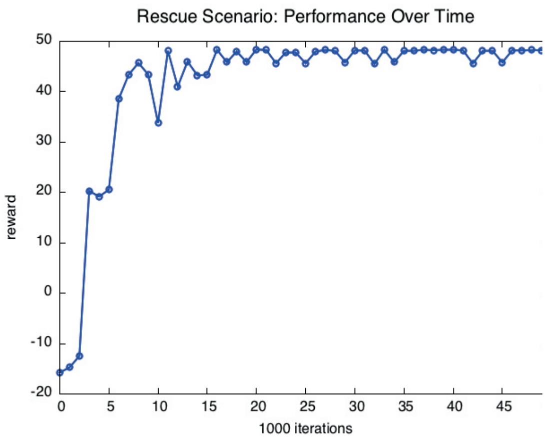
Ludwig-Maximilians-Universität München

Matthias Hölzl
hoelzl@pst.ifi.lmu.de

www.ascens-ist.eu/iliad

Some behaviors of ensembles operating in open-ended, changing environments cannot be completely specified during their design. Instead, the ensemble has to learn from experience and reason about novel situations as they arise. Using this approach, initial behaviors are learned by the ensemble from simulations of likely environments during design time; at runtime these behaviors are improved as the ensemble learns how its actual environment differs from expectations.

Iliad is a framework for this style of learning and reasoning. It supports deep learning and hierarchical reinforcement learning, predicate-logic reasoning with integrated support for constraint processing, inference in Bayesian networks, and heuristic planning.



Iliad's input language is called Poem. In Poem, programmers can leave choices of actions or values partially unspecified and indicate which learning or reasoning mechanisms should resolve the non-determinism of each choice. Therefore developers can either establish fixed behaviors, indicate design-time preferences or simply state the possible actions. Iliad will optimize these choices either by reasoning or by learning from feedback provided by the environment. Given sufficient knowledge or training, the actions determined by Iliad will converge to those with the highest expected value for the environment in which the ensemble is operating.

For example, in the rescue scenario, robots may combine map-based navigation planning with learning and reasoning about the environment to update inaccurate design-time assumptions with data gained at run time. This often leads to significantly better initial performance when compared to solutions solely based on learning; but even if the initial knowledge is highly inaccurate the learning mechanism will cause the behavior to improve as the robots obtain information about their environment at run time. In the depicted example, the design data was randomly initialized; in spite of this the performance of the system approached the optimal performance after a relatively short time.

Iliad can either be used as a knowledge repository for SCEL or as a stand-alone reasoner. Poem code can also be generated from executable parts of SOTA/GEM specifications written in the Prototype Verification System (PVS) specification language.

Monitoring, Awareness and Self-Adaptation

SPL - Stochastic Performance Logic

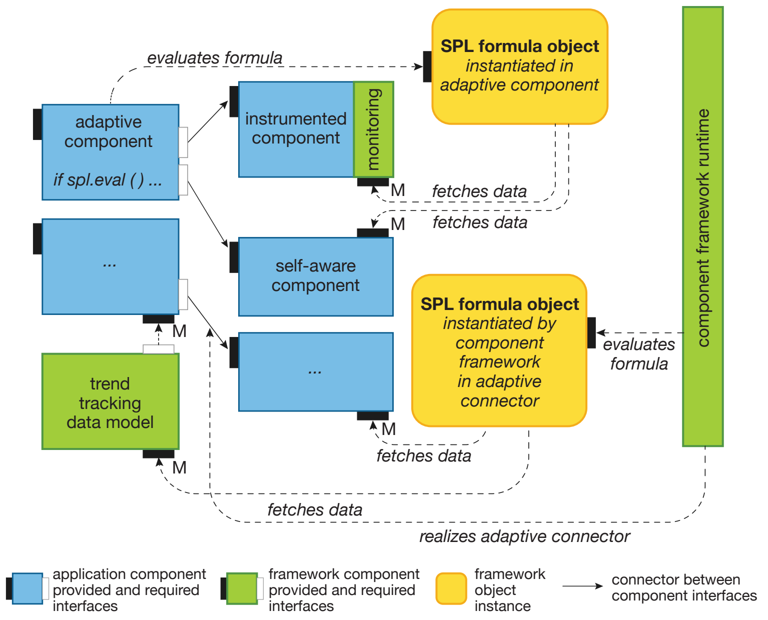
Charles University of Prague

Petr Tůma
petr.tuma@d3s.mff.cuni.cz

www.ascens-ist.eu/spl

SPL is a many-sorted first-order logic designed to reason about observed performance using intuitive expressions such as "A is faster than B on workload X". The logic relies on a sample-based interpretation, where the individual performance relations are evaluated using statistical hypothesis testing on collected measurements. The relations are evaluated in the context of a particular workload, parametrized by settings such as input data size.

Both declarative annotations and explicit evaluation are used to connect the logical formulas to code in application development and execution - as is the case in the examples of adaptive components and connectors. In both cases, code is associated with expressions that capture assumptions about performance and therefore help interpret the observed performance in the light of the available adaptation steps. These steps can include code adaptation identified by the IRM as part of the standard application execution, as well as manual code adaptation that the IRM delegates back to the ensemble development process.


evaluates formula
SPL formula object
instantiated in adaptive component
adaptive component
if spl.eval ( ) ...
instrumented component
monitoring
M
fetches data
M
fetches data
component framework runtime
...
self-aware component
M
SPL formula object
instantiated by component framework in adaptive connector
evaluates formula
...
trend tracking data model
M
fetches data
M
fetches data
realizes adaptive connector
application component provided and required interfaces
framework component provided and required interfaces
framework object instance
connector between component interfaces

SPL is implemented in prototypes that use automated code instrumentation through the Domain-Specific Language for Bytecode Instrumentation tool (DISL) to monitor the application. These prototypes demonstrate the use of the formalism in the testing domain, where assumptions about the application and environment performance made in the early stages of the development life cycle can be checked through testing.

Self-Awareness Platform

Università degli Studi di Modena e Reggio Emilia

Nicola Bicocchi
nicola.bicocchi@unimore.it

www.ascens-ist.eu/sap

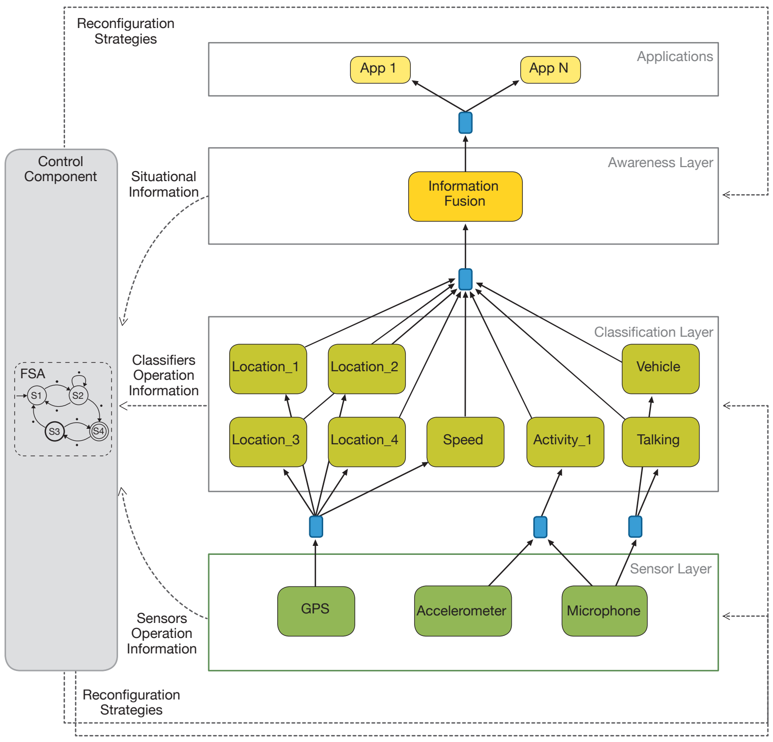
The Self-Awareness Platform is a self-aware and reconfigurable architecture for context awareness. It is a Java-based, lightweight and self-* framework whose purpose is to simplify experimental prototyping of context-aware applications to better understand the whole concept of context-awareness and its applicability in ubiquitous computing. In particular, programmers are allowed to completely decouple application logic from context understanding.

The framework allows speedy selection of data sources, connecting them to general purpose classifiers, and providing applications with contextual information encoded in a structured way.

Furthermore, the framework is highly dynamic and reconfigurable; modules can be loaded, unloaded and reconfigured at runtime using state-based automata. Specifically, depending on their function they can be hosted in three different layers, namely sensor, classification and awareness layers, and provide structured labels to applications.

Experiments have shown that the approach is effective in improving:

- energy efficiency on constrained devices
- classification precision and recall in several applications
- improving software engineering of pervasive applications



AVis - Awareness Visualizer Plug-in

Fokus - Fraunhofer Gesellschaft

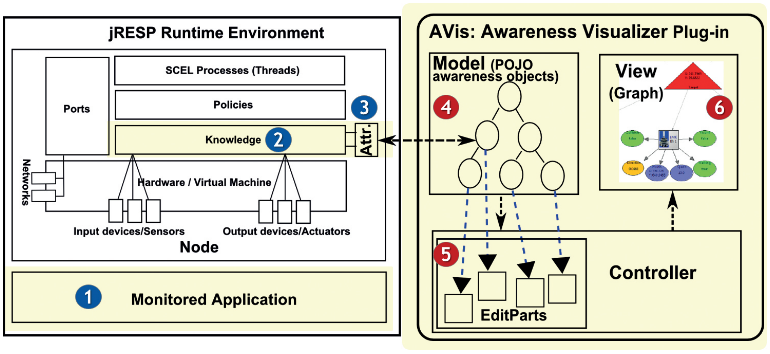
Dhaminda Abeywickrama
dhaminda.abeywickrama@gmail.com

www.ascens-ist.eu/avis

The Awareness Visualizer plug-in facilitates

- the monitoring of changes to awareness data of an autonomic system executed in the jRESP runtime environment
- the visualization of adaptation at runtime using a graph-like representation

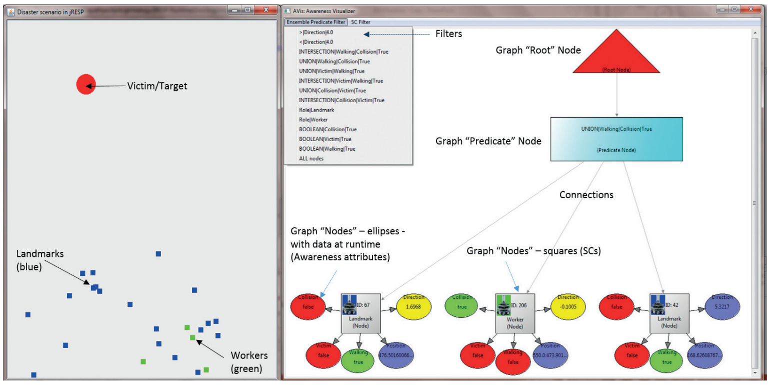
A key benefit here is to provide feedback to the engineer about the behavior of the complex awareness mechanism used, thus helping the decision-making process. This feedback can also improve any offline activities on the redesign of the system, verification and redeployment.



The AVis plug-in has been implemented as an Eclipse rich client application with GEF capabilities to support visualization. There are three main components in the plug-in – Model, View and Controller.

The Model is the data portion of the plug-in containing plain old java objects (POJOs) created for the monitored awareness attributes. These are instantiated at runtime using the knowledge attributes in the interface of a node in jRESP. The Observer-observable pattern in Java is employed for listening and notifying the state of the POJO awareness objects in the AVis plug-in when the corresponding state of the attributes in the node's interface is updated.

The View part draws the graph where the model is represented as nodes and connections in the graph. The Controller binds the model to the view, and for this, it listens for model changes and updates the view.


Disaster scenario in jRESP
Victim/Target
Landmarks (blue)
Workers (green)
AVis: Awareness Visualizer
Ensemble Predicate Filter
SC Filter
Filters
Graph "Root" Node
(Root Node)
Graph "Predicate" Node
UNION|Walking|Collision|True
(Predicate Node)
Connections
Graph "Nodes" – ellipses - with data at runtime (Awareness attributes)
Graph "Nodes" – squares (SCs)

```
>|Direction|4.0
<|Direction|4.0
INTERSECTION|Walking|Collision|True
UNION|Walking|Collision|True
UNION|Victim|Walking|True
INTERSECTION|Victim|Walking|True
UNION|Collision|Victim|True
INTERSECTION|Collision|Victim|True
Role|Landmark
Role|Worker
BOOLEAN|Collision|True
BOOLEAN|Victim|True
BOOLEAN|Walking|True
ALL nodes
```

Disaster scenario in jRESP (left) and the AVis plug-in (right).

Engineering Ensembles

EDLC - Ensembles Development Life Cycle

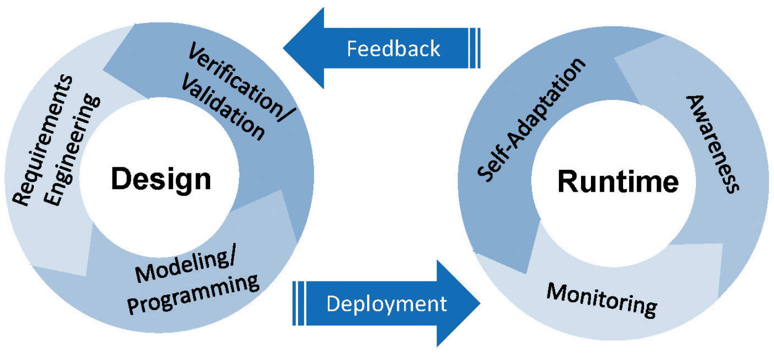
Ludwig-Maximilians-Universität München

Nora Koch
kochn@pst.ifi.lmu.de

www.ascens-ist.eu/edlc

The EDLC is a conceptual framework for autonomic systems that goes beyond addressing the classical phases of software development (like requirements elicitation, implementation and testing) as it also tackles aspects such as self-awareness, self-adaptation and self-expression. Such properties have to be considered from the early design phases to capture how the system should be adapted and how the system and environment should be observed in order to make awareness and adaptation possible at runtime.

Design activities comprise requirements engineering, modeling & programming, and verification & validation phases, which are performed offline. Runtime issues focus on activities performed online, such as monitoring, awareness and self-adaptation of ensemble-based software systems. Software deployment and the system feedback provide the connections between the online and offline activities.



The life cycle is represented as a “double-wheel” with two “arrows” between the wheels, providing three different feedback control loops:

- at design time which enables continuous improvement of models and code using the results of verification and validation and allowing for changing requirements;
- at runtime that implements self-adaptation based on awareness about the system and its environment;
- from the runtime back to improve the design with mechanisms that change architectural models and specification according to the runtime behavior of the continuously evolving system.

All **ascens** methods, tools and platforms focus at least on one EDLC phase, some of them on two or more, and several can be used in a tool chain – complementing each other – for the development of self-aware and self-adaptive systems.

DEECo - Dependable Emergent Ensembles of Components

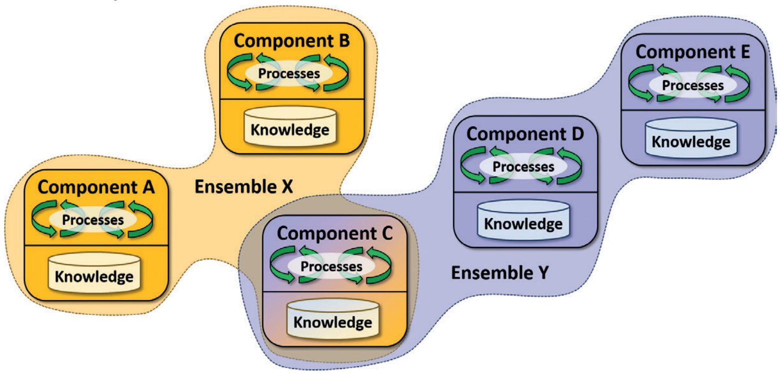
Charles University of Prague

Tomáš Bureš
bures@d3s.mff.cuni.cz

www.ascens-ist.eu/deeco

DEECo is a component framework where both component and ensemble are first class concepts. Each component constitutes state, referred to as knowledge, and behavior, expressed in terms of a set of processes. To achieve component interaction, components are dynamically composed into ensembles.

Membership of a component in an ensemble is declaratively expressed in terms of the membership condition, defined on component interfaces, which provide a partial view of component knowledge. Members of an ensemble interact in terms of implicit knowledge exchange, which is handled by the execution environment.



In DEECo, a component operates autonomously, based solely on its own knowledge, which is implicitly updated depending on the ensembles the component belongs to at a particular moment. A single component can be involved in several ensembles at the same time.

In order to bring the DEECo concepts closer to regular software development for evaluation and experimentation, its execution environment (jDEECo) has been created in Java. The DEECo concepts (components and ensembles) are mapped to Java via an internal domain-specific language realized by Java annotations. The core responsibility of the runtime environment – execution of knowledge exchange – is implemented in two ways: via distributed tuple-space middleware or via periodic broadcast, supporting deployment on mobile ad-hoc networks (MANETs).

```
component Vehicle features AvailabilityAggregator:
  knowledge:
    position = GPS(...),
    calendar = [ POI(WORKPLACE, 9AM−1PM), POI(MALL, 2PM−3PM), ... ],
    plan = { route = ROUTE(...), isFeasible = TRUE }
  process computePlan: in plan.isFeasible, in calendar, inout plan.route
    function: if (!plan.isFeasible) plan.route ← planner(calendar)
    scheduling: periodic( 5000ms )
  ...
ensemble UpdateAvailabilityInformation:
  coordinator: AvailabilityAggregator
  member: AvailabilityAwareParkingLot
  membership:
    ∃ poi ∈ coordinator.calendar: isClose(member.position, poi.position) && isAvailable(poi)
  knowledge exchange: coordinator.availabilities ← { (m.id, m.availability) | m ∈ members }
  scheduling: periodic( 2500ms )
```

Helena Modeling Approach

Ludwig-Maximilians-Universität München

Annabelle Klarl
klarl@pst.ifi.lmu.de

www.ascens-ist.eu/helena

The Helena approach is a formal modeling technique for ensembles that is centered around the notion of roles. Ensembles are built on top of a component-based platform as goal-oriented communicating groups of components. The functionality of each group is described in terms of roles which a component may dynamically adopt. Therefore, components can freely join and leave ensembles without breaking the overall functionality of the collaboration as long as another component takes over the abandoned role.

Components also dynamically adapt to new situations by changing their roles or by concurrently playing several roles (maybe in different ensembles) at the same time.

For example to retrieve a file from a peer-to-peer network storing files, such an ensemble is dynamically formed on top of the basic peer-to-peer platform. Peers need to adopt three different roles to first request the address of a provider from the network (via routing the request through the network) and then to request the file directly from the provider.

Ensembles can be executed with our Java framework jHelena, which transfers the role concept to object-oriented programming and follows the rigorous semantics of Helena models. Helena models can be described either graphically in a UML-like notation or in a domain-specific language fully integrated into the Eclipse Development Environment. A code generator transforms those models to jHelena code for execution.

Pattern Catalog

All project partners

Matthias Hölzl
hoelzl@pst.ifi.lmu.de

www.ascens-ist.eu/patterns

ascens provides a wealth of results for all phases of the development process. These research results are published in the scientific literature and documented in the **ascens** user guides and tutorials. However, for developers not intimately familiar with the entire body of the project's work it may sometimes be difficult to see which techniques are appropriate for a concrete development problem they are facing, or even that a technique addressing their specific problem exists.

Our catalog of design and development patterns addresses various problems faced by developers, gives solutions based on techniques developed by the **ascens** project, or sometimes well-known solutions that are applicable to **ascens** topics but where the applicability may not be obvious, and details the trade-offs that the solution implies.

The pattern catalog allows developers to easily find and evaluate which techniques are appropriate for their specific design situation.

To simplify the use of the pattern catalog, we have developed the **ascens** Pattern Explorer (Apex), a web application that allows full-text search across all pattern descriptions in addition to links between patterns.

Architectural Adaptation Patterns

Università degli Studi di Modena e Reggio Emilia

Franco Zanbonelli
franco.zambonelli@unimore.it

Mariachiara Puviani
mariachiara.puviani@unimore.it

www.ascens-ist.eu/adapatt

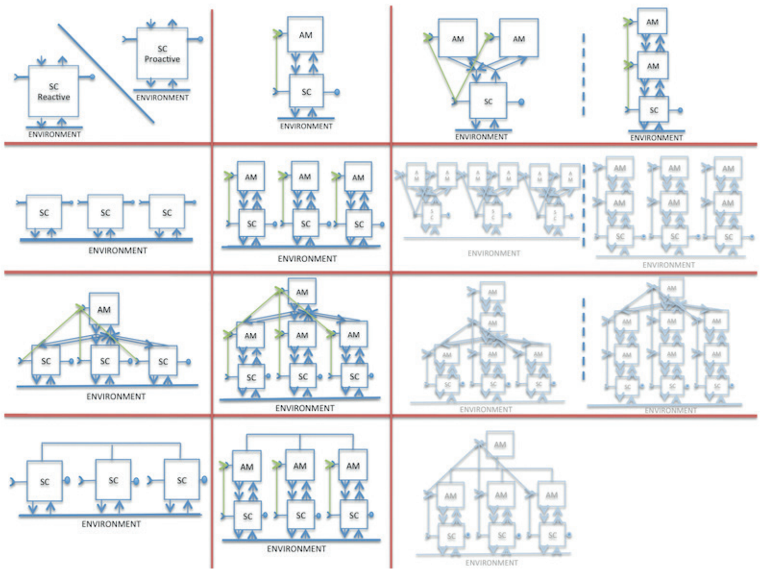
Patterns are able to describe generic solutions for a recurring design problem, so their use in designing self-adaptive systems is very relevant. Moreover, an adaptive architectural pattern is a conceptual scheme that describes a specific adaptation mechanism: it specifies how the component/system architecture can express adaptation.

Feedback loops provide the generic mechanism for adaptation and make it possible to create flexible runtime solutions. It consists of the continuous monitoring of the system and the application of corrections in order to approach the goal. Moreover it allows a system to become self-aware with respect to the quality of its operations, and pro-active if should there be any problems.

When developing an intelligent distributed system that needs to be adaptive, the use of an appropriate pattern that will enact adaptivity will help developers in their work. The pattern permits the developer to be guided to make the system exhibit a required behavior, even when unexpected situations occur. Moreover, a very important task for developing an appropriately performing self-adaptive system is to understand which pattern to choose.

The following taxonomy table arranges adaptation patterns in different levels:

- the single component level – first row;
- the ensemble level where the environment is considered as the means of adaptation (e.g., a bio-inspired system) – second row;
- the ensemble level where adaptation is delegated to an external agent called an Autonomic Manager (AM) that manages all the other agents (e.g., a centralized system with regard to the adaptation aspects) – third row;
- the ensemble level where adaptation is delegated to the agents themselves though their direct communication of adaptation mechanisms – fourth row.



SASOE - Design Guidelines and Strategy for Engineering Emergence

Università degli Studi di Modena e Reggio Emilia

Victor Noël
victor.noel@irit.fr

Franco Zambonelli
franco.zambonelli@unimore.it

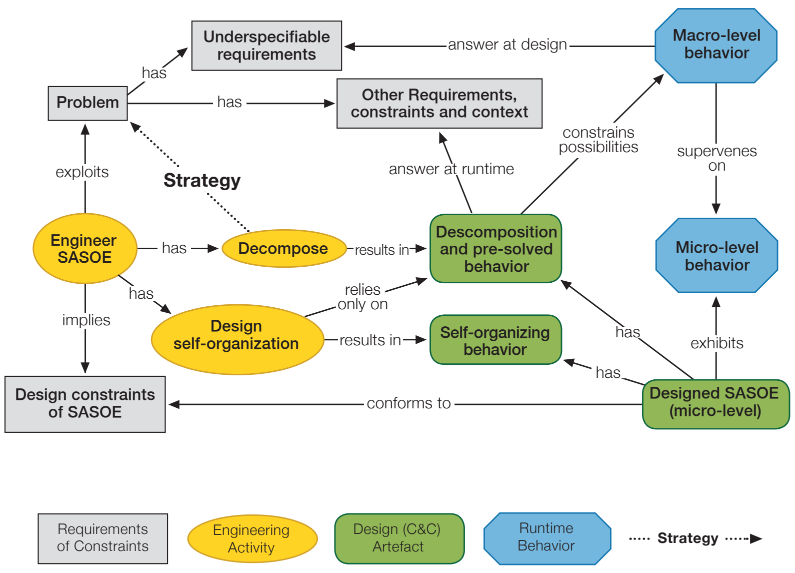
www.ascens-ist.eu/sasoe

These guidelines and strategy aim to guide engineers designing self-adaptive, self-organising ensembles exhibiting emergence (SASOE), as for example in the victims rescuing scenario of the **ascens** swarm robotic case study.

Engineering support is provided at the architectural level by rationalizing and explaining how the decomposition of an ensemble in components is related to the problem that has to be solved. In practice, this is useful for documenting and understanding the design choices made by the engineer; but more interestingly, this can be used as a strategy to follow in order to make such choices. The main idea is to exploit the description and the organization of the problem to be solved (i.e., the requirements to answer and in which context) to drive the decomposition.

Using self-organization – to handle the complexity and dynamics of the system environment and requirements – is a choice that is made at the very beginning of the development. This implies that the design activity must focus at the local micro-level of the components without prejudging how the actual global macro-level function of the ensemble is realized at runtime, making it emergent from an engineering point-of-view.

The first design activity is to decompose the system by choosing the components and how they relate to each other: the impact of this choice is of high importance for the successful functioning of the complete system. This decomposition activity is usually done in an non-rationalized ad-hoc way: in response to that, we propose these guidelines and strategy in order to enable engineers to produce complex software of quality.



Case Studies

Ludwig-Maximilians-Universität München

Philip Mayer
mayer@pst.ifi.lmu.de

Zimory GmbH

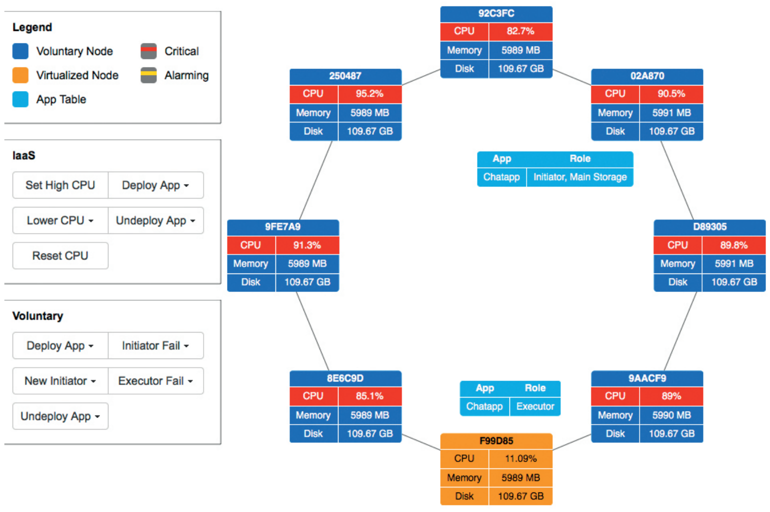
José Velasco
velasco@zimory.com

www.ascens-ist.eu/cloud

A Decentralized and Resilient Autonomic Cloud

The cloud case study of **ascens** realizes a decentralized and resilient autonomic cloud – a distributed software system able to execute applications in the presence of a permanently changing network and node infrastructure and thus difficulties such as leaving and joining computers, fluctuating usage, and applications with different requirements.

For the realization of this vision, **ascens** integrates cloud computing with voluntary computing and peer-to-peer computing and relies on nodes which are autonomic in their operation. In particular, nodes are self-aware of changes in load (arising either from cloud applications or from local applications external to the cloud) and in the network structure (i.e., nodes coming and going), which calls for self-healing properties.



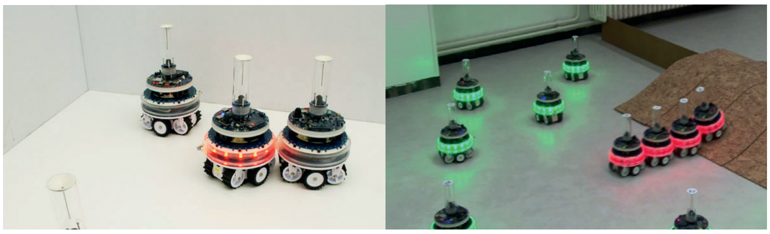
To prevent data loss in cases where nodes drop out of the system, redundant data storage has been added. Finally, executing applications in such an environment requires a fail-over solution or self-adaptation of the cloud to provide application execution resilience.

The **ascens** methods applied to the “autonomic cloud” endows clouds with a more dynamic and open functionality while at the same time maintaining the key benefits of a cloud as a reliable and flexible approach for using third-party resources and services.

The case study is realized in a prototype implementation called the “Science Cloud Platform” (SCP). The SCP is able to run on both physical and virtual hosts; the latter by using an Infrastructure-as-a-Service (IaaS) system such as the Zimory Cloud.

Collective Swarm Behaviors

Large multi-robot systems ("robot ensembles" or "robot swarms") have the potential of displaying desirable properties, such as robustness to individual failures through redundancy, and enhanced performance through parallelism and cooperation. Realizing such potential is challenging because of the lack of sound design methodologies. The aim of the robotics case study is to apply the methods developed by the partners of the **ascens** project to validate them and obtain novel, more robust behaviors for robot ensembles.



Université Libre de Bruxelles

Marco Dorigo
mdorigo@ulb.ac.be

École Polytechnique Fédérale de Lausanne

Francesco Mondada
francesco.mondada@epfl.ch

www.ascens-ist.eu/swarms

ARGoS - Autonomous Robots Go Swarming

ARGoS is an efficient, flexible, and accurate physics-based simulator for large robot swarms. The third version of ARGoS is one of the results of **ascens**. ARGoS is designed to be accurate, scalable, and flexible.

ARGoS can simulate thousands of robots in real-time on an average computer. Its architecture is multi-threaded and completely modular. Among its unique features, in ARGoS it is possible to partition the virtual space into regions managed by different physics engines running in parallel. ARGoS is open source software.

Université Libre de Bruxelles

Carlo Pinciroli
carlo@pinciroli.net

www.ascens-ist.eu/argos

Magnetic Gripper for Marxbot Modular Robot

For the swarm robotics cases study, a new robot module called "magnetic gripper" has been developed. This module is designed to allow a robot ensemble to manipulate objects and build structures. To facilitate grasping, the gripper is based on a magnetic switch that enables and disables a magnetic field.

The module has 3 degrees of freedom (lifting, tilting, and rotating with respect to the robot base). The module has sensing capabilities through 1 force sensor on the arm, 10 distance sensors, 10 proximity sensors and 2 microphones.

Mobsya

Michael Bonani
michael.bonani@mobsya.org

www.ascens-ist.eu/marxbot

Ensembles of Cooperative E-Vehicles

Volkswagen AG

Henry-Paul Bensler
henry.bensler@volkswagen.de

Nicklas Hoch
nicklas.hoch@volkswagen.de

www.ascens-ist.eu/
evehicles

Individual motorized mobility encompasses drivers, vehicles and infrastructure entities such as car parks, charging stations and roads. In **ascens**, these entities are modelled as autonomic components which have the ability to intelligently pursue a personal objective and to interact with other components in order to achieve group-level objectives.

A decentralized approach for the coordination of these autonomic components gives rise to important software design challenges, including (1) distributed reasoning design and (2) a system architecture to efficiently handle knowledge distribution and manage different belief states between the components.

ascens methods and tools (like simSOTA, SCEL, IRM) address these software design challenges, thereby enabling an efficient coordination of vehicles and infrastructure entities in a decentralized manner. The **ascens** approach focuses on self-adaptation capabilities of the vehicles in order to discretely handle failures and provide a seamless travel experience to the customer.

Runtime Simulation

Charles University of Prague

Tomáš Bureš
bures@d3s.mff.cuni.cz

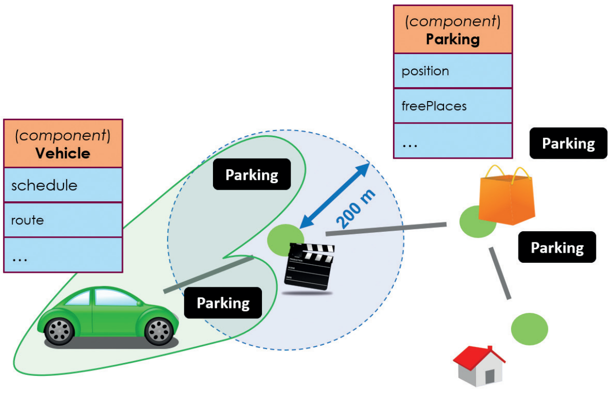
Volkswagen AG

Nicklas Hoch
nicklas.hoch@volkswagen.de

www.ascens-ist.eu/sim

The e-Mobility case study uses the jDEECo environment to monitor system states and handle self-awareness and self-adaptation actions during system runtime. The jDEECo runtime environment employs DEECo components, such as the "vehicle" component, and DEECo ensembles such as the vehicle-PLCS (Parking Lot Charging Station) ensemble, which optimizes the parking choices of an ensemble of vehicles in reference to the capacity usage of the car parks.

JDEECo embeds the Multi-Agent Transport Simulation (MATSim), which is an execution environment capturing the physical interaction of drivers, vehicles and infrastructure components.



HSCSP - Hierarchical Soft Constraint Satisfaction Problems

Università di Pisa

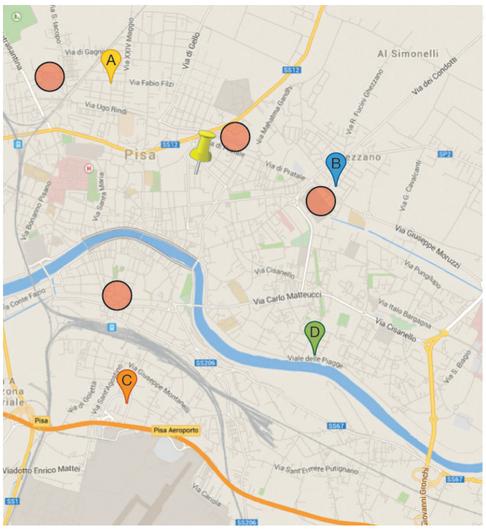
Ugo Montanari
ugo@di.unipi.it

www.ascens-ist.eu/hscsp

In the **ascens** e-mobility case study, the parking allocation problem consists of finding the best parking lot for each vehicle. However, a globally optimal solution may be very expensive to find. For this reason, two approaches have been proposed, based on the coordination of declarative and procedural knowledge.

The first approach consists of decomposing the original optimization problem into local problems concerning its components. Each of them is solved separately, using the soft constraint logic programming framework. A coordination strategy, implemented by an orchestrator, is then used to verify that the local solutions satisfy all global constraints. If this is not so, parameters of the declarative implementation are modified and a new feasible (not necessarily optimal) global solution is computed.

A demonstrator has been implemented, using the CIAO logic programming system for the declarative part, Java for the orchestrator part, and the Google map system (for Pisa) to represent the maps, to compute the best paths and to implement the visual interface.



All the vehicles would like to find the best parking lots.

In particular the vehicle's user A must reach 📌.

Vehicles	A B C D
Parking lots	●
Appointment location of A	📌

The second approach consists of representing soft constraint satisfaction problems as terms of an algebraic specification, similar to a process calculus. These terms are then inductively evaluated in a domain of cost functions, where operators are interpreted as optimization steps.

Using a dynamic programming approach, optimization is then carried out on these functions. The declarative aspects include heuristic solutions of the secondary optimization problems and lower dimensional approximations of the intermediate tables.

Application of Methods, Languages & Tools to Case Studies

All project partners

Nikola Šerbedžija

nikola.serbedzija@fokus.
fraunhofer.de

www.ascens-ist.eu/tools

The **ascens** project developed methods, languages and tools for all phases of the ensemble development life cycle (EDLC). A rich problem space of swarm robotics, cloud computing and e-mobility application domains was used to test and refine the theoretical concepts in pragmatic settings.

In the final project stage, all three major **ascens** cases studies were developed and deployed according to the EDLC methodology. The following table provides a cross-reference of the use of **ascens** methods, languages and tools for each of the case studies and ensemble development life cycle phases. The table is particularly useful for identifying which tools could be used in specific phases of the EDLC.

EDLC Phase	Robotics	Autonomic Cloud	e-Mobility
Requirements Engineering	SOTA (2) Gem (3) Poem (18)	ARE (2) IRM (3)	IRM (3) simSOTA (29)
Modeling & Programming	SCEL (4) jRESP (5) FACPL (6-7) AIA / MAIA (9) MESSI (10) Poem / Iliad (18) ARGoS (28) SASOE (26)	Adapt. Patterns (25) Helena Roles (24) SCEL (4) FACPL (6-7) NCPi (11) KnowLang Policies (16)	SCEL (4) HSCSP (30)
Verification & Validation	SMC-BIP (14) jRESP Analysis (5) Comp. Verif. (14) MultiVeStA (15) MESSI (10)	jRESP Analysis (5)	jDEECo (23)
Deployment	ARGoS (28) Marxbot Magnetic Gripper (28)	SPL (19) SCP (27) Zimory IaaS (27)	jDEECo (23) MATSim / Routeplanner (29)
Monitoring	ARGoS (28)	SCP (27) Zimory IaaS (27) SPL (19)	jDEECo (23) DISL / SPL (19) MatSim / Visualization (29)
Awareness	Poem (18) AVIs Plug-in (21)	SCP (27)	jDEECo (23) SPL (19)
Self-Adaptation	ARGoS (28) AVIs Plug-in (21) Iliad / Poem (18)	Zimory IaaS (27) SCP (27)	jDEECo (23) IRM (3)
Feedback	Poem (18)	SPL (19)	MATSim (29)

(#): brochure page reference

ascens in numbers

	1.10.2010 31.03.2015
Partners and Third Parties	
Universities	11
Research organizations	4
Companies	2
Participating countries	7
Participants	
Researchers	81
Associated researchers	21
PhD students	20
Web presence	
Web pages	75
Blog entries	30
Links to **ascens** project site	56
Publications	311
Books/ Proceedings	6
Book contributions	13
Articles in journals	58
Papers in conferences and workshops	215
Technical reports	19
Joint publications	109
Best paper awards	8
Presentations and tutorials	154
Software prototypes/products	27
Courses (based on project results)	57
Events	
Summer Schools (co-organized)	7
ascens Spring School 2015	1
Conferences/Workshops (co-organized)	100
Fairs/Exhibitions (ICT 2013, CeBIT 2015)	2

All project partners

Nora Koch
kochn@pst.ifi.lmu.de

www.ascens-ist.eu

ascens stand - ICT 2013 - Vilnius, Lithuania

Authors

Dhaminda Abeywickrama

Saddek Bensalem

Henry Bensler

Nicola Biococchi

Michael Bonani

Roberto Bruni

Tomáš Bureš

Jacques Combaz

Andrea Corradini

Rocco De Nicola

Marco Dorigo

Mike Hinchey

Nicklas Hoch

Matthias Hölzl

Annabelle Klarl

Nora Koch

Jan Kofron

Diego Latella

Michele Loreti

Andrea Margheri

Philip Mayer

Francesco Mondada

Ugo Montanari

Victor Noël

Carlo Pinciroli

Rosario Pugliese

Mariachiara Puviani

Matteo Sammartino

NIkola Šerbedžija

Francesco Tiezzi

Petr Tůma

Andrea Vandin

Emil Vassev

José Velasco

Martin Wirsing

Franco Zambonelli

ascens

autonomic service-component ensembles

www.ascens-ist.eu

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 257414. This project is part of the Future and Emerging Technologies Proactive Initiative.

Partners	Ludwig-Maximilians-Universität München • Università di Pisa • Università degli Studi di Firenze • Fraunhofer Gesellschaft • UJF-Verimag • Università degli Studi di Modena e Reggio Emilia • Université Libre de Bruxelles • École Polytechnique Fédérale de Lausanne • Volkswagen AG • Lero - University of Limerick • Zimory GmbH • IMT Advanced Studies Lucca • Mobsya • Charles University of Prague
Third Parties	Istituto di Scienza e Tecnologie della Informazione "A. Faedo" • National Center for Scientific Research • Institut Polytechnique de Grenoble
Project coordinator	Ludwig-Maximilians-Universität München Institute for Informatics Prof. Dr. Martin Wirsing info@ascens-ist.eu Tel +49 89 2180 9154 Fax +49 89 2180 9175
Work package leaders	Rocco De Nicola • Roberto Bruni • Mike Hinchey • Franco Zambonelli • Saddek Bensalem • Petr Tůma • Nicola Šerbedžija • Matthias Hölzl • Nora Koch • Martin Wirsing
Published by	**ascens** Consortium www.ascens-ist.eu
Text editor	Nora Koch kochn@pst.ifi.lmu.de
Layout	vpdesigns.de info@vpdesigns.de