www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D8.2: Second Report on WP8

The ASCENS Service Component Repository (first version)

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **Version 2.2 (30.7.2011)**

Lead contractor for deliverable: **LMU**
Author(s): **Matthias Hölzl, Lenz Belzner, Thomas Gabor, Annabelle Klarl (LMU)**

Reporting Period: **2**
Period covered: **October 1, 2011 to September 30, 2012**
Submission date: **November 12, 2012**
Revision: **Final**
Classification: **PU**

Project coordinator: **Martin Wirsing (LMU)**
Tel: **+49 89 2180 9154**
Fax: **+49 89 2180 9175**
E-mail: **wirsing@lmu.de**

Partners: **LMU, UNIPI, UDF, Fraunhofer, UJF-Verimag, UNIMORE, ULB, EPFL, VW, Zimory, UL, IMT, Mobsya, CUNI**

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

Work package 8 contains three tasks. Task T8.1, “Challenges of Developing SCEs in the Real World” was completed after month 6 of the project. Tasks T8.2, “A Service-Component repository for self-aware autonomic ensembles” and T8.3, “Best Practices for SCEs” were started in months 13 and 19, respectively, and are ongoing.

This deliverable reports on Task T8.2. We present the requirements for the service-component repository (SCR), the design choices we made for its implementation and the current state of the web application for the SCR. In addition we give an overview of the service components that are included in the SCR and discuss those service components that are not detailed in other deliverables.

Contents

1 Introduction

Work Package 8 contains three tasks: T8.1, "Challenges of Developing SCEs in the Real World" identified the main challenges involved in developing ensembles and presented them in a structured format. This task was active during the first six month of ASCENS; it was completed in month 12 of the project with the delivery of deliverable D8.1 [Höl11]. The other two tasks started in month 13 and 19 of the project, respectively, and are still ongoing.

The goal of Task T8.2, "A Service-Component repository for self-aware autonomic ensembles" is described in D8.1 as "[...] produce a ready-to-use repository of key service components to be used in all kinds of ensembles. We will make this catalogue available online to ensure widespread dissemination." T8.3, "Best Practices for SCEs" has the objective "to codify best practices discovered during the project in a form that is easily accessible for SCE practitioners. This work will result in a catalogue of SCE patterns for the overall results of the ASCENS project." In accordance with the Description of Work this deliverable reports on task T8.2, in particular on the specification and development of the infrastructure for the service-component repository (SCR) and on the components in the SCR. Deliverable D8.3, to be completed at the end of the third reporting period, will report on task T8.3.

1.1 Deviations from the Description of Work

The Description of Work [ASC11] mentions the possible inclusion of an ASSL reasoner in the components of the repository. Contrary to our expectations at the time, the ASSL reasoner will not be made available by its copyright holders under a license that permits its inclusion into the SCR. However, ASSL is not used in the project and has largely been superseded by KnowLang, therefore this change has no impact on the research efforts of the project or the usefulness of the SCR. The KnowLang tools are being developed as part of the ASCENS project and will be included in the SCR.

1.2 Relationship to Other Work Packages

As an integrative work package, the goal of WP8 is to integrate the results of WPs 1–7 into a coherent approach to engineering ensembles: "...the foundational work done of ASCENS, applied to the case studies in work package 7 must be put into a larger perspective from an engineering point of view, thereby identifying best practices for service component ensembles. Thus, this work package is integrative in nature by combining results from the foundational work packages and the real-world case studies investigated in ASCENS." [ASC11]. Therefore, the work of WP8 is strongly influenced by the results of WPs 1–7.

For the SCR this can be seen in Sect. 3: the SCR already contains components developed as part of all other technical work packages. The ensemble-engineering approach currently under development in WP8 is heavily influenced by the technical results from WPs 1–5 and the availability of the SDE and its integrated tools in WP6. The scenarios for the development approach are taken from WP7 and the best practices are being extracted from work performed on the case studies.

1.3 Structure of the Deliverable

The following section reports on the requirements for the ASCENS service-component repository, the design choices made for the development of the resulting web application, and the current state of the SCR application. Sect. 3 gives an overview of the service-components included in the SCR, contains pointers to those components in the SCR detailed in other deliverables, and describes those components which are not the focus of another current deliverable.

2 The Service-Component Repository

According to the Description of Work, T8.2, "A Service-Component repository for self-aware autonomic ensembles" is intended to "produce a ready-to-use repository of key service components to be used in all kinds of ensembles. We will make this catalogue available online to ensure widespread dissemination." In this section we describe first the requirements for the SCR, then its implementation.

2.1 Requirements for the SCR

The direct requirements that the DoW specifies for the repository itself are few and can be summarized as follows: The SCR should

- provide a catalogue of key service components
- support users of the SCs in the repository, not their developers
- be made available online to ensure widespread dissemination

The contents of this repository should consist of key SCs that are useful in multiple domains. The original intent of the SCR was to have a catalogue for human consumption that contains a relatively modest number of medium-sized to large components. Note that it was (and is) not a goal of the SCR to provide the complete infrastructure for developing the included components. There exist many online services (e.g., *Github*, *Google Code* or *Sourceforge*) that provide functionalities such as revision control, issue trackers, or hosting of project web sites, and many component developers are accustomed to one of these services or have already set up their own internal system. Trying to duplicate the functionality of these systems would require a large expenditure of resources and be unlikely to provide any measurable benefit. The SCR is focused on discovering components, a functionality which is not provided by other services.

During the work on tasks T8.1 and T8.3 it became clear that for ensembles the distinction between design time, deployment and run-time is much less clear than for traditional systems, and that many of the tasks that are traditionally performed by human designers during the development process might also be performed autonomously by SCs while they are deployed and operating in their real execution environment. In particular, a goal-based SC might discover at run-time that it cannot reach the desired (achieve) goal, for example because a service on which it relies in its current strategy is no longer available. An adaptation strategy might query some service to find out whether a feasible correction to the current strategy is available that does not rely on the missing service, e.g., by using a different sevice that can provide the same or similar results. The functionality that has to be provided by such a service is similar to that of the SCR, but the available services have to be in a more formalized manner to allow autonomic discovery of service components by other components. To support these kinds of goal-based behaviors, we have augmented the initial, relatively modest, requirements for the SCR with the following more ambitious ones to support autonomous discovery of (relatively fine-grained) components:

- Extend the human-readable description of components with a formal specification of their properties
- Provide a service-oriented interface to permit automated discovery of components

There are many proposals for formally describing components and services. For example, in the area of web services, the existing languages range from the (relatively) simple, mostly name-based Web Service Description Language (WSDL) [W3Cc] to expressive specification languages based on

first- or higher-order logic such as the Semantic Web Services Language and Ontology SWSL/SWSO [W3Ca, W3Cb] and their underlying process model PSL [BG05]. In the ASCENS project, the KnowLang specification model [VHM+12] defines probabilistic policies controlling a partially-observable Markov decision process (POMDP). None of these specification styles is appropriate for all domains, and the best practices developed in Task T8.3 should include guidelines about making the right choices between expressivity and efficiency. Threrfore we obtain an additional requirement:

- Provide an expressive language for the formal specification of components so that the desired specification styles can all be expressed in a single language

Being able to use different specification styles in the same instance of the SCR not only allows various domains with different requirements for their specifications to co-exist, it will also enable us to experiment with various strategies to specify, model and design service components for the development of the guidelines and patterns for Task T8.3.

2.2 Design Choices

The initial version of the SCR consisted of a set of wiki pages in a structured format. For each service component in the catalogue the following data was provided:

- *Name:* The name of the component
- *Domain:* The domain to which the component belongs, e.g., general for general-purpose components, cloud for components belonging to the cloud case study, or robotics for components mainly useful for robotics applications
- *Tags:* A list of tags that describe the component, for example location service for components that provide location-based services
- *Status:* The current development status of the component, e.g.: pre-alpha, alpha, beta, or stable
- *Source:* URL(s) that provide more information, downloads of binary and source packages, etc.
- *Partner:* The ASCENS partner(s) that produced the component
- *Contact:* The person(s) to contact for questions on the component, including contact information
- *Platform and Infrastructure:* The infrastructure the component uses or the platform that the component resides on. May include programming languages (C++, Java), operating system (Linux, OSX, Windows), frameworks, libraries, and tools, including versions if applicable
- *Description:* A description of the purpose and possible uses of the component

While this solution satisfies the first set of requirements, it is clear that such a system cannot accomodate the additional goals presented in the previous section. We have therefore designed a second version of the SCR based on the following design:

- Augment the previously described data model with one or more formal specifications for each component to allow discovery based on goals and the effects that a component has on the environment
- Persist the data for service components in a data base

- Specify a domain-description language to express effects of service-components on their environment and a query language that can be used to search for components that, e.g., can be used to reach a specified goal given a set of preconditions
- Provide reasoning services that can execute queries in the query language against the data stored in the database
- Provide a service-based back end that exposes all functions of the SCR, so that the SCR can potentially be accessed from a web-based front end, the ASCENS SDE, or components of an ensemble
- Implement a web-based front end to satisfy the requirement for online access

The most challenging design problem in this list are the specification of the languages and reasoning engine. Since ASCENS defines a family of expressive knowledge-representation and modelling languages [DHL+12], using one of them for the SCR is a natural choice. SOTA and GEM are mathematical models without underlying implementations, therefore they are not suitable as languages for the SCR. SCEL is mostly focused on executable behaviors and not on goal-based reasoning, therefore it is not the most expedient choice as language for the SCR. Both KnowLang and POEM are appropriate as domain-description and query language. We decided to base the SCR on POEM and use *Iliad* (see Sect. 3.3) as the reasoning service for the SCR. *Iliad*, the POEM interpreter developed as part of Task T8.3, supports reasoning about full first-order and restricted higher-order theories, as well as more efficient but also more limited constraint-based reasoning. The *Iliad* implementation uses the Snark theorem prover [Sti] as its core (first-order) reasoning mechanism. Snark itself is well suited for integration into the SCR for a number of reasons already demonstrated in other projects [Wal01, DHM+01]:

- It is optimized for reasoning over large theories and permits far-reaching control over its inference process in order to achieve acceptable performance
- Snark has a powerful sort theory that can further restrict the proof size based on ontological (taxonomic) information specified about the domain
- Using procedural attachments it is easy to integrate database queries for components or specialized search methods into the reasoning process

The *Iliad* implementation provides additional features that are useful for the SCR:

- *Iliad* facilitates the specification of domain knowledge, e.g., by generating domain-closure or unique names axioms
- It is possible to specify *Iliad* strategies to restrict the search space for queries; this can lead to potentially large reductions of the search space when compared to unrestricted search
- *Iliad* uses Snark's built-in facilities for constructive proof search to generate answers from queries; if Snark cannot find a constructive proof, *Iliad* extracts constraints from the failed proof attempts and uses constraint-solving techniques to arrive at solutions

Thus, by using POEM as the specification language for service components in the SCR and *Iliad* as the reasoning engine, we obtain an expressive specification language for components into which many of the simpler languages can easily be translated, and we can provide efficient reasoning over service-component specification. While this design is not as efficient as a solution based on a special-purpose reasoner for a dedicated service-description language, it allows us to easily compare various styles

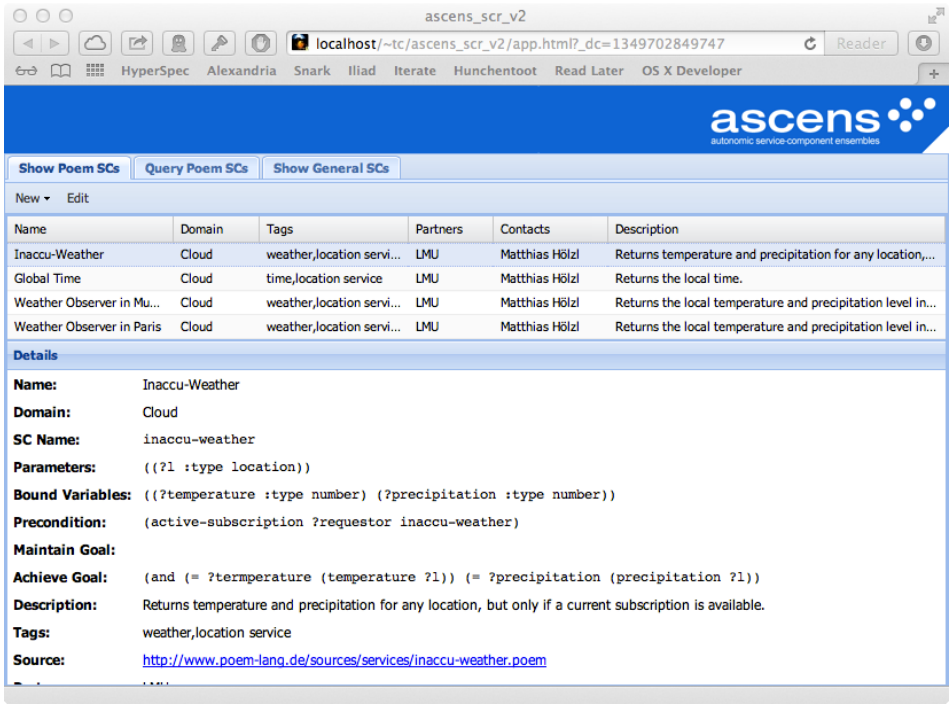

Figure 1: A screenshot of the service-component repository

of representing behaviors of components. Furthermore, we can perform experiments to compare the practical impact of various choices, e.g., complete versus incomplete reasoning or the impact that component specifications with different expressivity have on the development process.

The first prototype of this design was completed at the end of the second reporting period and is described in the next section.

2.3 Implementation

The SCR is implemented as a three-tiered, service-based application. The data tier is provided by an Apache CouchDB [Pro] database that stores a document for each service component in the repository. The decision to use a schema-less, document-centric database was made to simplify maintaining a catalogue containing different styles of SC descriptions and to easily allow the addition of several formal specifications to a component.

The logic tier exposes a Representational State Transfer (REST)-based service interface [FT00] that allows create, read, update and delete (CRUD) operations on individual service components, retrieval of all components, as well as logical queries. Queries can contain a `domain` parameter that both specifies a logical theory in which the query is evaluated and restricts the set of components to those from the selected domain (and general-purpose components available in every domain). Internally the logic tier synchronizes updates to component descriptions with corresponding changes to the logic theory of the *Iliad* system; furthermore it extracts the necessary information to pass back to clients from the substitutions returned as result of *Iliad* proofs. To avoid another marshalling/unmarshalling step, the logic tier is, like the *Iliad* system, implemented in Common Lisp; the REST interface uses the Hunchentoot web server [Wei] for request processing.

The presentation tier consists of a web-based interface that allows users to access the SCR interactively. A screenshot of the main page can be seen in Fig. 1. The interface contains two tabs that allow the user to browse lists of all components for which formal specifications have been provided (“Show

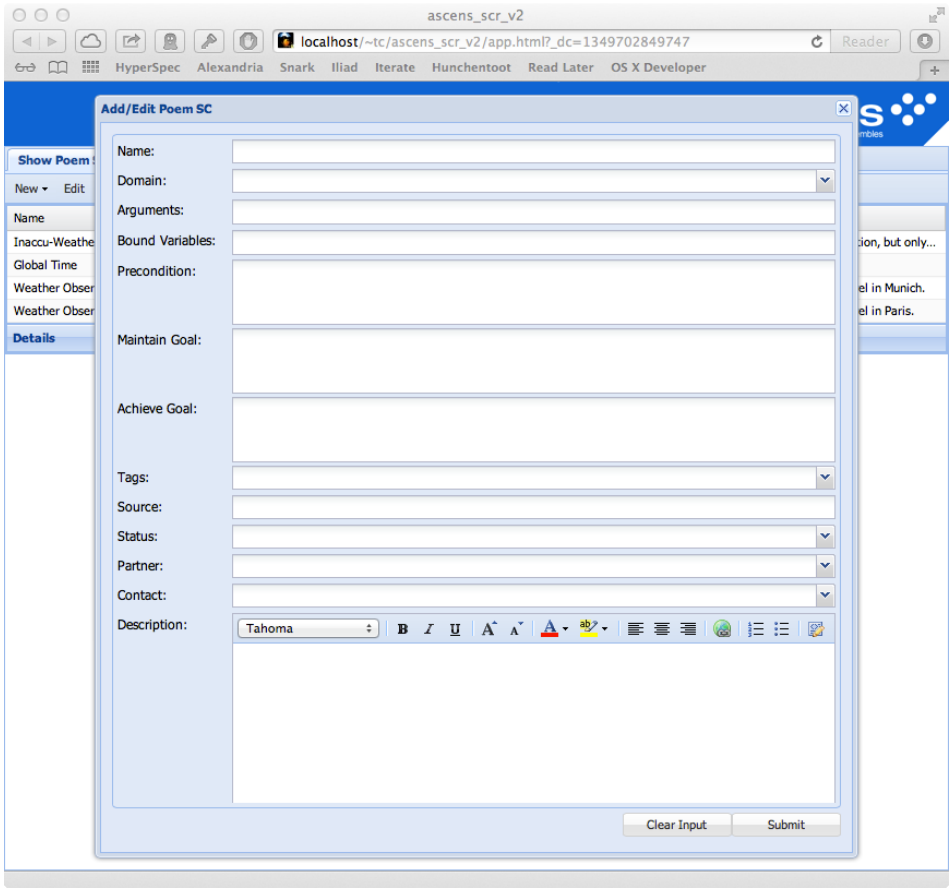

Figure 2: Adding a new service component

Poem SCs") and of all components for which no formal specification is available ("Show General SCs"); a third tab allows the user to issue queries for components. Below each list of SCs is a pane that shows detailed information about a selected component. As can be seen from the screenshots, the user interface is geared toward a SOTA-style form of specification where explicit preconditions, maintain-goals (utilities in SOTA) and achieve goals are defined for each component. This choice also permits us to easily add POEM strategies to the SCR since they are based on the same kind of specification. Conversely, each component specified in this manner can be used as a strategy in POEM models.

The presentation tier allows users of the SCR to perform CRUD operations on individual components; as an example, the interface for adding a new SC to the repository is shown in Fig. 2. The interface to the query engine is currently very simple (see Fig. 3): It allows the user to enter a query term and returns a list of all service components found by the query and displays them in a similar manner to the other panes. This interface is very restricted: the service components are determined as the values of all substitutions for a variable of sort `service-component`; therefore each query has to contain exactly one such variable. Additional substitutions returned by the query are not explicitly shown in the current interface, however their values are substituted into the query results in which they appear. Furthermore, it is not possible to use the current interface to perform other types of queries that are allowed by the back end, e.g., checks whether a certain composition of components satisfies a desired property. We expect to enhance the interface to the query functionality as we gain more experience with using the SCR in the development of new service components.

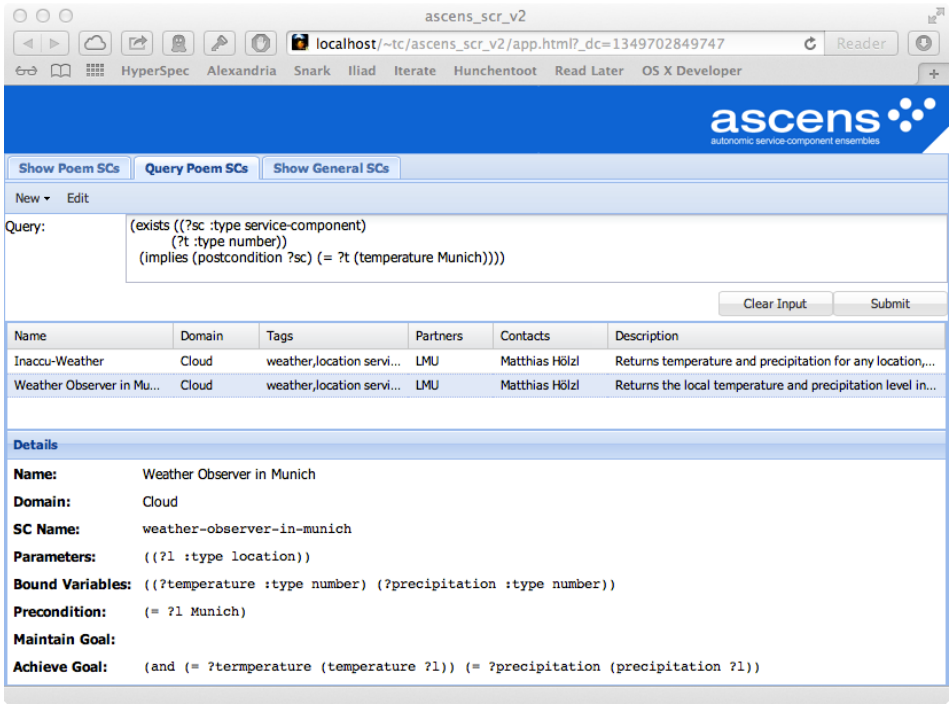

Figure 3: Querying for a service component

3 Service Components

The infrastructure of the SCR described in the previous section needs to be populated with service components in order to be useful. The Description of Work states: "Among others, we expect such components to include a constraint-based planner, an ASSL reasoner, a knowledge-processing-engine component, and control-loop components." We have already stated in Sectbe . 1.1 that it will not be possible to include the existing ASSL reasoner due to licensing problems, but that its functionality will be subsumed by the KnowLang toolkit which will be included in the SCR. The constraint-based planner is already available as part of the *Iliad* implementation of the POEM language (see Sect. 3.3), the knowledge-processing-engine component is being developed as part of the KnowLang implementation (see D3.2 [VHM+12]). A simulator for feedback loops and self-adaptive behavior is already available (see D4.2 [ZAC+12]).

In this section we provide an overview of the medium-sized and large service components included in the SCR. Most of them are detailed in other deliverables; to avoid redundancy we only provide a pointer to their description. Note that there is also a certain overlap between the components reported in this deliverable and the software development tools presented in deliverable D6.2 since, e.g., simulators are included in both deliverables.

3.1 SCs Reported in Other Deliverables

- The jRESP runtime environment for SCEL-based programs is reported in Deliverable D1.5 [BGeHy+12].

- The DEECo component model and the corresponding jDEECo runtime are reported in Deliverable D1.5 [BGeHy+12].

- The KnowLang tools and reasoner provide knowledge represenation and reasoning services; they are described in Deliverable D3.2 [VHM+12].
- The simulation tool for feedback loops and self-adaptive patterns developed in work package 4 is described in Sect. 4.2 of Deliverable D4.2 [ZAC+12].
- The jSAM Eclipse plugin integrates a set of tools for stochastic analysis of concurrent and distributed systems, specified using process algebras and is described in more detail in Deliverable D6.2 [CHK+12].
- The BIP compiler is used to generate code from BIP programs. The generated code is compiled and linked with an execution engine which is used to schedule the execution of the BIP model. The BIP system can be used to compute either single execution traces or all possible execution traces. A more detailed description is given in D6.2.
- The Gimple Model Checker (GMC) is an explicit-state code model checker for C/C++ programs. GMC is able to discover low-level programming errors such as buffer overflows, memory leaks, null-pointer dereferrence. It is described in D6.2.
- The Maude Daemon Wrapper embeds the Maude framework into the Eclipse environment and thus into the SDE. A more detailed description is given in D6.2.
- SPL is a Java framework for implementing application adaptation based on observed or predicted application performance. It is described in D6.2.
- The ARGoS simulator provides a high-fidelity simulation environment for swarm robotics. While it is most frequently used to test the implementation of robot controllers, it can also be employed as component in an ensemble, e.g. to simulate various strategies before committing to a course of events in an negotiate-commit-execute cycle. An overview of ARGoS is given in Deliverable D6.2.
- The Science Cloud platform is formed by individual instances, or clients, running on individual machines, each of which is an SC. The Science Cloud platform is described in Deliverable D7.2 [ŠMP+12].

Table 1 summarizes the medium to large components in the SCR. The following sections contain short descriptions of those service components that are not described in other Deliverables. All components in the SCR were either developed or significantly improved as part of ASCENS.

3.2 ARGoS-Lua and Lua-Tools

Lua-Tools is a collection of tools for the development of adaptive systems in the Lua programming language which is particularly suited as an embedded language or for use on small devices. ARGoS-Lua is a library that permits the developments of robot controllers for ARGoS in Lua.

3.2.1 Lua-Tools

Lua [IdFF11] is a scripting language that is frequently used in embedded devices because it is small, portable, and easy to interface to libraries written in C. The small size also means that by default Lua misses many features or libraries that facilitate the development of large systems in other programming languages. However the powerful meta-programming features of Lua permit the users to provide these facilities as libraries. Lua-Tools is such a library that focuses on the development of

Name	Description	Deliverable
ARGoS	Simulator for robot swarms	D6.2
ARGoS-Lua	Lua wrapper for the ARGoS simulator	D8.2
BIP	The BIP compiler suite and Engine	D6.2
GMC	The Gimple Model Checker	D6.2
Iliad	Analysis and execution of Poem specifications	D8.2
jDEECo	Runtime for the DEECo component model	D1.5
jResp	Runtime environment for SCEL	D1.5
jSAM	Stochastic analysis of systems specified as process algebras	D6.2
KLT	KnowLang Toolset: Representation and reasoning for knowledge bases	D3.2
Lua-Tools	Components for the implementation of self-adaptive systems in Lua	D8.2
MDW	Maude Daemon Wrapper: Integration of Maude into the SDE	D6.2
MESS	Tool for implementing and analyzing self-assembling strategies	D8.2
SCP	Nodes of the Science Cloud Platform	D7.1
SPL	A Java framework for implementing application adaptation based on observed or predicted performance	D6.2
—	Simulation tool for feedback loops and self-adaptive behavior	D4.2

Table 1: Service components currently registered in the SCR

adaptive, autonomous systems. Lua-Tools contains modules containing general utilities, an object system for Lua, message-based communication as well as probabilistic search, soft-constraint solving and evolutionary algorithms.

The general utilities provide facilities for serializing and de-serializing arbitrary object graphs, tools for manipulating Lua tables, and support for non-deterministic program flow. The latter allow the exploration of several alternative computations without persistent changes to the global or local state of the application.

Lua does not provide native support for object oriented programming. Using the very versatile data structures Lua provides (mainly tables and lambdas with lexical binding), OO can easily be built into Lua, though. In fact, the standard introduction to Lua [Ier06] already illustrates two different approaches to implementing objects in Lua. However, this flexibility has led to a large number of non-interoperable object systems used in various Lua libraries. Therefore Lua-Tools provides an abstract module to provide the functionality needed to write code in an object oriented manner, without committing to a specific implementation of object orientation. This allows the use of various existing object systems using a unified interface. In addition the module oo itself offers three different implementations of object systems that can be chosen on the fly.

The modules for message-based communication provide `qry`, `get` and `put` primitives for exchanging data between processes in the same or different address spaces (using the 0mq message queue [Zer] to transport data between processes).

The final set of modules allows the user to solve arbitrary soft constraints specified in the formalism of [HMW09] using stochastic search. The module `constraints` allows the evaluation and management of constraints, with the module `domains` providing several default search domains. The module `evolution` builds on top of that and provides a simple yet flexible implementation of an evolutionary search algorithm. Likewise, the module `sa` is able to perform simulated annealing to find solutions in suitable search spaces.

Together these modules provide a versatile set of components for the development of lightweight, adaptive systems.

3.2.2 The ARGoS-Lua Library

The ARGoS-Lua library allows users to program controllers for ARGoS [PTO+11] in Lua instead of C++. This simplifies the development of controllers by shielding the programmer from many complexities of C++, and it simplifies the prototyping of controllers since controllers in ARGoS-Lua can be modified while the simulator is running, whereas controllers written in C++ require a restart of ARGoS, as well as recompilation and relinking of the controller for each change. As an example, the main part of a controller performing a random walk with obstacle avoidance is given in Fig. 4.

The controller shown in Fig. 4 implements a straightforward algorithm and does not use the Lua-Tools library in its implementation. For more complex adaptive behaviors, we expect the combination of Lua-Tools and ARGoS-Lua to greatly simplify the implementation.

3.3 *Iliad*—Implementation of Logical Inference for Adaptive Devices

Iliad is an implementation of the POEM language [HBKK12] that allows developers to analyze and execute POEM specifications. *Iliad* consists of three core components:

- A reasoner that provides a mechanism for performing logical inference and computing substitutions satisfying a logical formula.

```
function control_step ()
   -- Read sensor values and compute movement
   local proximity_sensor = state.sensors.proximity
   local readings = proximity_sensor:get_readings()
   local size = readings:size()
   local acc = argos.Vector2(0, 0)
   for i = 0, size - 1 do
      local reading = readings[i];
      acc = acc + argos.Vector2(reading.value, reading.angle)
   end
   acc = acc / size
   local angle = acc:angle()

   -- Control the movement
   local wheel_actuator = state.actuators.wheels
   local v = state.velocity
   if math.abs(angle:get_value()) < 0.1 and acc:length() < 0.1 then
      wheel_actuator:set_linear_velocity(v, v)
   else
      if angle:get_value() < 0 then
         wheel_actuator:set_linear_velocity(0, v)
      else
         wheel_actuator:set_linear_velocity(v, 0)
      end
   end

   -- Set the LEDs.
   local leds = state.actuators.leds
   leds:set_all_colors(argos.Color_BLACK)
   angle:unsigned_normalize()
   local led_index = angle / argos.Radians_TWO_PI * 12
   leds:set_single_color(led_index, argos.Color_RED)
end
```

Figure 4: An ARGoS-Lua controller for random walk with obstacle avoidance

- A byte-code interpreter that implements executable strategies. The bytecode interpreter is closely integrated with the reasoner, so that strategies can use the reasoner to infer facts about the state of the world or to choose applicable components.

- A compiler that translates POEM specifications into logical theories for the reasoner and executable strategies running on the byte-code interpreter. The compiler also ensures that consistency is maintained between the logical theory and more conventional object-oriented behavioral specifications possible in POEM.

In addition we plan to develop a set of (soft and hard) constraint solvers and optimizers as well as machine-learning components that can be integrated into the reasoning engine. Initial implementations of some of these components have already been started.

The *Iliad* system can be used throughout the development process:

- In the early stages of the development of a component or ensemble it can be used to precisely specify SOTA or GEM models and to reason about abstract properties based on their logical specification and a domain theory.

- The initial specification can then be refined by developing (partially) executable strategies and by detailing the domain model.

- Once the specification has been detailed, the POEM specification can serve as a high-level program that can be executed by *Iliad*; during the execution the reasoner can be invoked by strategies.

A more detailed description of the *Iliad* system will be given in Deliverable D8.3 in the next reporting period.

3.4 MESS—Maude Ensemble Strategies Simulator

The *Maude Ensemble Strategies Simulator (MESS)* is a tool for implementing and analyzing self-assembling strategies with Maude.

The observation underlying MESS is that adaptive self-assembling strategies are a crucial mechanism that allows groups of simple individual entities to act as a single complex entity exhibiting emergent behaviours. Notable examples for this include bacteria or insect swarms, modular and self-assembling robots and software components with dynamic coupling mechanisms.

MESS is a tool to implement and analyze these kind of self-assembling strategies that follows the recently proposed conceptual framework for adaptation [BCG+12]. It exploits the declarative and reflective features of the Maude language for the implementation and relies on the Maude tool framework for the analysis of the specified system.

MESS can be used to prototype, simulate and analyze self-assembling strategies in the early development phases. Thereby errors in self-assembly strategies (e.g., SCs with incorrect assumptions about the state of the environment) can be discovered and corrected in the early stages of the development process.

4 Ongoing Work and Plan for Year 3

The main activities of tasks T8.2 and T8.3 will continue throughout the third reporting period. In the following sections we detail the work planned for the next 12 months.

4.1 Task T8.2

We will use the current version of the SCR in the development of solutions for the case studies and continue its development based on the experiences gained during this process. Currently we envision the following improvements for the third reporting period:

- *Security for public web access:* As a publicly visible server, the SCR has to be protected against malicious queries. While the *Iliad* runtime provides options to limit the resource consumption of queries, they are relatively easy to circumvent with carefully crafted queries. More importantly; *Iliad* provides facilities to access arbitrary code on the underlying platform during queries. Before allowing public access to the SCR it will be necessary to provide robust mechanism to protect the host computer from malicious queries; this will be our highest priority in the development of the SCR, so that a publicly available version of the SCR can be provided within the first three months of the third reporting period.
- *Multiple domains and specifications:* Currently each service component in the SCR is restricted to a single domain and its underlying theory. Future extensions should allow SCs to belong to different domains, and potentially have several specifications conforming to different domain theories.
- *Improved query interface:* The current user interface to queries is rather restricted since it only allows queries that return a single list of components; despite this it is necessary for users to have a thorough understanding of POEM and the domain theories used to describe components. It is likely that experience with the SCR will lead to a redesign of the user interface that addresses these points.
- *Reasoning about compositions:* The SCR places no restrictions on the queries that can be performed; therefore support for automatic derivation of service-component compositions that can reach a goal that no individual component can reach on its own is possible. We will investigate which kinds of compositions can be computed efficiently and how the interface should support the development of service compositions.
- *Integration with KnowLang:* Once the KnowLang tools are sufficiently far developed, we will investigate the possibility of translating (a subset of) KnowLang into POEM so that specifications for service components in the SCR can be written in KnowLang.

4.2 Task T8.3

Several strands of development are ongoing in task T8.3:

- *Awareness Engineering:* We are currently investigating how the notion of "awareness" can be integrated into the development process, and how the required levels of awareness depend on the adaptation space in which the system is operating. We call this process "awareness engineering;" preliminary results are presented in the technical report [Höl12]. Awareness engineering will be integrated into the development approach based the DEECo component model [BGeHy$^+$12] for a unified approach to ensemble engineering.
- POEM*:* The POEM language has been developed to
  - support awareness engineering,
  - enable knowledge-intensive modeling approaches while also being suitable for domain in which models based on sophisticated knowledge-representation methods are not feasible,

  - allow a seamless transition from high-level SOTA, GEM and KnowLang models to executable SCEL specifications.

  As far as possible, POEM is designed to allow a modelling style that is close to conventional object-oriented approaches. In the third reporting period we will refine this way of modelling, improve the POEM language and the *Iliad* implementation and apply the approach to more challenging scenarios, in particular to the disaster recovery scenario described in deliverable D7.2 [ŠMP+12].

- *Pattern Catalogue:* We have started with the collection of patterns for developing self-aware, self-adaptive SCs and SCEs. We will continue to add new patterns and improve the existing patterns. In addition we will investigate ways to formalize patterns and integrate them with the SCR.

An important consideration during the third reporting period will be the integration and possible unification of the different strands of development in WP8.

5 Summary

This deliverable reports on the progress of Task T8.2 "A Service-Component repository for self-aware autonomic ensembles". During the cause of the project, the simple requirements initially given for the SCR were considerably enlarged to support the software development process developed as part of Task T8.3 "Best Practices for SCEs". The most important new requirement is support for fine-grained SCs with formally specified behavior. Therefore, the SCR is implemented as a web application incorporating the *Iliad* reasoning engine and allowing formal component specification using the POEM language. Several medium to large SCs have been catalogued in the SCR, research with fine-grained SCs is ongoing.

In the next year, work in WP8 will focus on ensemble engineering in general, and in particular on awareness engineering and its integration into other engineering approaches such as DEECo; the continuing development of the POEM language and its implementation; a catalogue of patterns for ensemble development, and the integration of the results in WP8 and other work packages.

References

[ASC11] ASCENS Project: Annex I—Description of Work, June 2011. Version 2.2.

[BCG+12] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In *15th International Conference on Fundamentals of Software Engineering (FASE'12)*, LNCS. Springer, 2012.

[BG05] Conrad Bock and Michael Gruninger. Psl: A semantic domain for flow models. *Software and Systems Modeling*, 4(2):209–231, 2005.

[BGeHy+12] Tomáš Bureš, Ilias Gerostathopoulos, Vojtěch Hork´ y, Jaroslav Keznikl, Jan Kofroň, Michele Loreti, and František Plášil. Deliverable D1.5: Language Extensions for Implementation- Level Conformance Checking, November 2012.

[CHK+12] Jacques Combaz, Vojtch Hork, Jan Kofro, Jaroslav Keznikl, Alberto Lluch Lafuente, Michele Loreti, Philip Mayer, Carlo Pinciroli, Petr Tma, and Andrea Vandin. Deliverable D6.2: The SCE Workbench and Integrated Tools, Pre-Release 1, November 2012.

[DHL+12] Rocco De Nicola, Matthias Hoelzl, Michele Loreti, Alberto Lluch Lafuente, Ugo Montanari, Emil Vassev, and Franco Zambonelli. Joint Deliverable JD2.1: Languages and Knowledge Models for Self-Awareness and Self-Expression—Self-Awareness, Self-Expression, Adaptation, November 2012.

[DHM+01] Grit Denker, Jerry R. Hobbs, David L. Martin, Srini Narayanan, and Richard J. Waldinger. Accessing information and services on the daml-enabled web. In *SemWeb*, 2001.

[FT00] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 407–416, New York, NY, USA, 2000. ACM.

[HBKK12] Matthias Hölzl, Lenz Belzner, Annabelle Klarl, and Christian Kroiss. The POEM Language. Technical Report 7, ASCENS, October 2012. `http://www.poem-lang.de/documentation/TR7.pdf`.

[HMW09] Matthias Hölzl, Max Meier, and Martin Wirsing. Which soft constraints do you prefer? *ENTCS*, 238(3):189–205, 2009.

[Höl11] Matthias Hölzl. Deliverable D8.1: First Report on WP8: Challenges of Developing SCEs in the Real World, November 2011.

[Höl12] Matthias Hölzl. Awareness Engineering. Technical Report 8, ASCENS, October 2012. `http://www.pst.ifi.lmu.de/~hoelzl/ascens/TR8.pdf`.

[IdFF11] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Passing a language through the eye of a needle. *Commun. ACM*, 54(7):38–43, 2011.

[Ier06] Roberto Ierusalimschy. *Programming in Lua (2. ed.)*. Lua.org, 2006.

[Pro] The Apache Project. CouchDB Web Site. `http://couchdb.apache.org`, last accessed October 2012.

[PTO+11] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Timothy Stirling, Álvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. Technical Report TR/IRIDIA/2011-009, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2011.

[ŠMP+12] Nikola Šerbedžija, Mieke Massink, Carlo Pinciroli, Manuele Brambilla, Diego Latella, Marco Dorigo, Mauro Birattari, Philip Mayer, José Angel Velasco, Nicklas Hoch, Henry P. Bensler, Dhaminda Abeywickrama, Jaroslav Keznikl, Ilias Gerostathopoulos, Tomas Bures, Rocco De Nicola, and Michele Loreti. Deliverable D7.2: Second Report on WP7. Ensemble Model Syntheses with Robot, Cloud Computing and e-Mobility, November 2012. ASCENS Deliverable.

[Sti] Mark E. Stickel. SNARK - SRI's New Automated Reasoning Kit. `http://www.ai.sri.com/~stickel/snark.html`, last accessed October 2012.

[VHM+12] Emil Vassev, Mike Hinchey, Ugo Montanari, Nicola Bicocchi, and Franco Zambonelli. Deliverable D3.2: Second Report on WP3. The KnowLang Framework for Knowledge Modeling for SCE Systems, October 2012.

[W3Ca] W3C Consortium. Semantic Web Services Language (SWSL). http://www.w3.org/Submission/SWSF-SWSL/, last accessed: October 2012.

[W3Cb] W3C Consortium. Semantic Web Services Ontology (SWSO). http://www.w3.org/Submission/SWSF-SWSO/, last accessed: October 2012.

[W3Cc] W3C Consortium. Web Service Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, last visited: October 2012.

[Wal01] Richard J. Waldinger. Web agents cooperating deductively. In *Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers*, FAABS '00, pages 250–262, London, UK, UK, 2001. Springer-Verlag.

[Wei] Edi Weitz. Hunchentoot - The Common Lisp web server formerly known as TBNL. http://weitz.de/hunchentoot/, last accessed October 2012.

[ZAC+12] Franco Zambonelli, Dhaminda B. Abeywickrama, Giacomo Cabri, Mariachiara Puviani, Matthias Hölzl, Andrea Corradini, Alberto Lluch Lafuente, and Rocco De Nicola. Deliverable D4.2: Second Report on WP4. Component- and Ensemble-level Self-Expression Patterns: Report on Experimental and Simulation Activities, and requirements for Tools Implementation, October 2012.

[Zer] The 0MQ web site. http://www.zeromq.org, last accessed October 2012.