www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D6.4: Fourth Report on WP6

The SCE Workbench and Integrated Tools, Final Release

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **Version 3.0 (29.4.2014)**

Lead contractor for deliverable: **CUNI**
Author(s): **D. Abeywickrama (UNIMORE), V. Horký, J. Kofroň, M. Kit (CUNI), A. Lluch Lafuente (IMT), M. Loreti, P. Mayer (LMU), P. Tůma (CUNI), A. Vandin (IMT)**

Reporting Period: **4**
Period covered: **October 1, 2013 to March 31, 2015**
Submission date: **March 12, 2015**
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

The progress of ASCENS workpackage WP6, which deals with tool development and integration, has been described annually in deliverables D6.1, D6.2 and D6.3. The deliverable D6.4 concludes this sequence with the description of progress in the final project year.

As a major departure from the previous tool deliverables, this year we do not present the overall tool landscape – instead, we focus on progress updates only. This is to avoid redundancy with the joint deliverable JD4.2, which provides complete tool landscape overview together with detailed description of each tool.

Contents

1 Introduction

The ASCENS project tackles the challenge of building systems that are open ended, highly parallel and massively distributed. Towards that goal, the ASCENS project considers designing systems as ensembles of adaptive components. Properly designed, such ensembles should operate reliably and predictably in open and changing environments. Among the outputs of the ASCENS project are methods and tools that address particular issues in designing the ensembles.

The structure of the ASCENS project reflects the multiplicity of issues in designing the ensembles. Separate workpackages aim at topics such as formal modeling of ensembles or the knowledge representation for awareness. The tool development activities of these individual workpackages are coordinated through WP6, a workpackage dedicated to keeping track of tool development and directing tool integration.

The progress of workpackage WP6 is reported in annual deliverables. The deliverable D6.1 collected the tool integration requirements. The deliverables D6.2 and D6.3 presented two preliminary tool releases. This deliverable, deliverable D6.4, coincides with the final tool release. Additionally, the joint deliverable JD4.2 provides a complete overview of the tools. To avoid duplicating information between D6.4 and JD4.2, D6.4 focuses on recent progress only (this is different from the past annual deliverables, which also provided a complete overview for reader context).

The goal of the tool release is to maximize the practical outreach beyond project scope – hence, effort has been made to have all tools as much self describing as possible, with the accompanying documentation in the usually preferable form of online help, examples and tutorials. The textual deliverables reference the online releases and inform about project progress, however, they are not meant to supplant the tool documentation.

As in the past project years, the workpackage collaboration relies especially on personal meetings and distributed development support. The personal meetings include the regular project meetings, where a dedicated slot is reserved for planning and coordinating the tool integration activities. In this reporting period, these were specifically the March general meeting in Modena and the July general meeting in Braunschweig. The regular meetings are complemented with bilateral partner meetings where more detailed issues are discussed.

In the final project year, most tool updates focused on supporting the case studies. The development activities also included the general bugfixing and documenting work, related to preparing the tools for final release. We present a quick summary focusing on workpackage collaboration, followed by a more detailed description for those tools where significant update activities took place.

- WP1 focuses on the languages for coordinating ensemble components, providing the foundation for work on the runtime environment for ensembles, in particular the jDEECo and jRESP frameworks. This past year, both frameworks were extended significantly – jDEECo with support for realistic network and realistic traffic simulation, jRESP with support for high level software development and integration with other project tools, jSAM and FACPL.

- WP2 focuses on the models for collaborative and competitive ensembles. Among the tool update activities, we have released the MAIA tool, implementing the Adaptable Interface Automata formalism in Maude.

- WP3 deals with knowledge modeling for ensembles. Here, the KnowLang tool updates included evolving some parts of the grammar compiler, and especially working on the KnowLang reasoner.

- WP4 activities concern the ensemble self expression, with modeling and simulation being prominent. With the ARGoS and SimSOTA platforms being fairly stable, the tool updates here include release of the AVis visualization tool.

- WP5 deals with the verification techniques for components and ensembles, among the tool updates in the last year is the work on the GMC model checker.

Together with WP7 and WP8, the WP6 workpackage forms an integrated block of activities focused on applying the project results. Where WP6 provides tool integration, WP7 drives the case studies that use the tools, and WP8 complements the tools with ensemble engineering practices.

2 Updates

The following sections describe major updates of each tool in the last reporting period, together with a brief refresh of the tool purpose. A complete list of tools is available in the joint deliverable JD4.2.

2.1 AVis

2.1.1 Tool Purpose

Monitoring in the EDLC is an activity performed at runtime to observe and collect awareness data of the system and environment to trace awareness and adaptation capabilities. The monitored awareness data can be a component's status (e.g. its current location) or information about the environment in which the components are executing (e.g. monitored sensor data), and adaptation is the runtime modification of the awareness data in a component's knowledge repository.

In this context, the Awareness Visualizer (AVis) is an Eclipse plug-in we have developed for tracing the awareness and adaptation capabilities of an application executing in the jRESP runtime environment. The AVis plug-in, which contains three main components (i.e. model, view and controller), has been developed as a rich client application with Graphical Editing Framework (GEF) capabilities.

2.1.2 Development Progress

The AVis plug-in was developed during the fourth year of the ASCENS project.

The AVis plug-in has been integrated with the jRESP runtime environment to facilitate the monitoring of changes to awareness data. Here, the monitored application (see step 1 in Fig. 1) can be any application scenario executing in jRESP. The model encompasses the data portion of the plug-in architecture, containing POJOs (Plain Old Java Objects) created for the monitored awareness attributes. These are created at runtime using the knowledge attributes in the interface of a node in jRESP. We employ the Observer-observable pattern in Java for listening and notifying the state of the POJO awareness objects in our visualizer plug-in when the corresponding state of the attributes in the node's interface are updated (see 3–4, Fig. 1).

The AVis plug-in has been assessed using two scenarios of the Swarm robotics case study. They are the use of different robot types for different roles (disaster scenario with landmark and worker robots), and the use of one robot type for several different roles (e.g. explorer, rescuer, help rescuer, low battery).

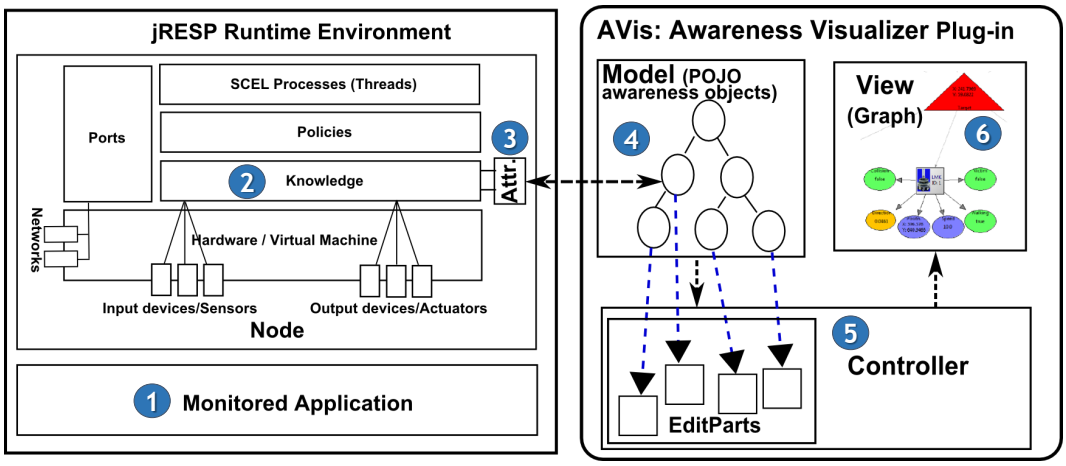

Figure 1: AVis plug-in system architecture and jRESP.

2.2 FACPL: Policy IDE and Evaluation Library

2.2.1 Tool Purpose

FACPL [MMPT13] is a tiny policy language for writing policies and requests. It has a mathematically defined semantics and can be used to regulate interaction and adaptation of SCEL components. FACPL provides user-friendly, uniform, and comprehensive linguistic abstractions for policing various aspects of system behaviour, as e.g. access control, resource usage, and adaptation. The result of a request evaluation is an authorisation decision (e.g. permit or deny), which may also include some obligations, i.e. additional actions to be executed for enforcing the decision.

FACPL is supported by an IDE and a Java implementation library. The IDE facilitates writing the desired policies in FACPL syntax, with features such as code completion and syntax checks. The IDE automatically produces a set of Java classes implementing the FACPL code, using the specification classes defined in the FACPL library. The library, according to the rules defining the language semantics, implements the request evaluation process, given as input a set of Java-translated policies and the request to evaluate.

2.2.2 Development Progress

The FACPL library and IDE have been developed during the third year of the project. In the fourth year of the project, the Java evaluation environment has been integrated within the jRESP runtime environment, thus enabling a full evaluation of the policy layer when programming ensembles using SCEL. Furthermore, the FACPL policy syntax has been added to the HL-SCEL tool, which supports the development of SCEL applications using standard programming constructs, such as conditional branches and loops, and an automatic generation of jRESP code.

2.3 Gimple Model Checker

2.3.1 Tool Purpose

Gimple Model Checker (GMC) is an explicit-state code model checker for C and C++ programs. It can reveal errors manifesting themselves in particular (usually rare) interleavings which are hard to find via testing. GMC supports multi-threaded programs and executes all possible interleavings to discover errors manifested only in certain thread schedules. From the ASCENS project perspective,

GMC is unique in that it can check some ensemble related properties, such as particular sequences of accesses to the ensemble knowledge (using custom assertion statements in the code).

On the technical side, GMC detects low-level programming errors such as invalid memory usage (buffer overflows, memory leaks, use-after-free defects, uninitialized memory reads), null-pointer dereferences, and assertion violations. GMC understands not only the pthread library [pth], but also offers means to add support for other thread libraries based on the same principles.

2.3.2 Development Progress

During the last project period, the work on GMC especially focused on implementing missing parts of the C++ runtime as provided by the GCC tool chain. Specifically, this included support for built-in synchronization as well as initialization routines, which were crucial for running (and verifying) multi-threaded programs. We also discovered several issues arising when using the standard template library (STL) in verified programs, caused by very frequent use of template nesting.

We also finished the integration of GMC into SDE, which is now available in the form of an Eclipse update site and enables the GMC to be used during everyday C/C++ development directly from SDE (and generally any Eclipse installation containing the C/C++ development support).

2.4 MAIA

2.4.1 Tool Purpose

As part of a research line pursued in collaboration between project partners, we have presented an essential model of adaptable transition systems [BCG$^+$12a] inspired by white-box approaches to adaptation [BCG$^+$12b] and based on foundational models of component based systems [dAH01, dA03]. The key feature of adaptable transition systems are control propositions, a subset of the atomic propositions labelling the states of our transition systems, imposing a clear separation between ordinary, functional behaviours and adaptive ones. Interestingly, control propositions can be exploited in the specification and analysis of adaptive systems, focusing on various notions proposed in the literature, like adaptability, control loops, and control synthesis. We instantiated our approach on Interface Automata (IA) [dAH01, dA03], yielding Adaptable Interface Automata (AIA) [BCG$^+$12a].

MAIA is an implementation of AIAs in Maude, allowing one to specify AIAs, to draw them, and to perform operations on them such as product, composition, decomposition and control synthesis.

2.4.2 Development Progress

MAIA has been developed during the third year of the project and released in the fourth year. It currently comes as a set of Maude files to be imported by the specifications of adaptable interface automata modelling adaptive systems.

2.5 Iliad

2.5.1 Tool Purpose

Iliad is a framework for building awareness mechanisms [HG15] for open-ended, distributed systems based on machine learning and reasoning techniques. 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.

Iliad is based on a flexible communication protocol called Hexameter. Hexameter implements the SCEL **get**, **qry** and **put** operators on top of the cross-platform, open source networking library ØMQ (`zeromq.org`). At the moment of writing Hexameter front-ends for Lua, Common Lisp, Java and JavaScript are available and can seamlessly interoperate. Therefore, Iliad can not only be used as a stand-alone reasoner but also as a knowledge repository for SCEL or as learning component for other reasoning systems such as the KnowLang reasoner.

2.5.2 Development Progress

During the last reporting period our focus was on simplifying the input language for Iliad, supporting learning for service components that do not synchronize their activities, and improving the scalability of the reinforcement learning system. These goals were achieved by implementing a new frontend for Iliad that is based on extended behavior trees (XBTs). This front-end allowed us to integrate hierarchical variants of lenient Q-learning [HG15] which are particularly well suited to uncoordinated learning in distributed settings, and the simple structure of XBTs improves scalability by allowing the integration of special-purpose planners and reasoners into a hierarchical learning framework.

2.6 Science Cloud Platform

2.6.1 Tool Purpose

The Science Cloud Platform (SCP) is the software system developed as part of the science cloud case study of ASCENS. The SCP is a platform-as-a-service cloud computing infrastructure which enables users to run applications while each individual node of the cloud is voluntarily provided (i.e., may come and go), data is stored redundantly, and applications are moved according to current load and availability of server resources.

The SCP can also take advantage of IaaS (Infrastructure-as-a-Service) platform such as the Zimory Cloud [Zim14], when available. In this case, new virtual machines running the SCP can be started on demand, and shut down to conserve energy when no longer needed.

The Science Cloud Platform serves as the main technical demonstrator for the cloud case study of ASCENS, integrating many of the newly researched methods and techniques into one software system.

2.6.2 Development Progress

This year, the work on SCP has focused on two areas, an overhaul of the communication strategy and the integration of support for an IaaS (Infrastructure-as-a-Service) platform.

Firstly, we have created an alternative implementation for communication on the application level with the integration of a gossip (endemic) strategy which uses dedicated roles at each node as specified in the Helena approach [KMH14]. This increases scalability since it does not depend on global broadcasts as in the ContractNET implementation, and serves to structure the implementation along role-based lines.

The second area concerns the integration of the Zimory Cloud [Zim14], an industry-strength IaaS solution. Each node in the system has been extended with the ability to interact with an IaaS solution

(when available). Thus, if no node is available for executing a certain application, a new virtual machine with the required capabilities is started on demand. Once online, the application is moved to this machine. If the application is later shut down or a non-virtualized machine becomes available, the virtual machine is shut down again, thus conserving energy.

The final SCP version thus contains all of the functionality envisioned in the requirements listed in the results of the first year of ASCENS while taking advantage of many of the research results which became available in subsequent years of the project.

2.7 SPL

2.7.1 Tool Purpose

SPL is a Java framework for implementing application adaptation based on observed or predicted application performance [BBH+12]. The framework is based on the Stochastic Performance Logic, a many-sorted first-order logic with inequality relations among performance observations. The logic allows to express assumptions about program performance and the purpose of the SPL framework is to give software developers an elegant way to use it to express rules controlling program adaptation.

The SPL framework internally consists of three parts that work together but can be (partially) used independently. The first part is a Java agent that instruments the application and collects performance data. The agent uses the Java instrumentation API [Ora12]; the actual byte code transformation is done using the DiSL framework [MZA+12]. The second part of the framework offers an API to access the collected data and evaluate SPL formulas. The third part of the framework implements the interface between the application and the SPL framework. This API is used for the actual adaptation.

2.7.2 Development Progress

The work in the fourth year of the project has focused on integrating SPL-based feedback into the development process. A prototype SPL tool version can now take the performance annotations and the workload implementation from performance tests, perform the required measurements and inject the results into standard software documentation as generated using Javadoc. The same mechanism can be used to display feedback from the ASCENS ensemble development lifecycle runtime in relevant code locations during development.

2.8 jDEECo: Java runtime environment for DEECo applications

2.8.1 Tool Purpose

jDEECo is a Java-based implementation of the DEECo component model [BGH+12] runtime framework. It allows for convenient management and execution of jDEECo components and ensemble knowledge exchange.

The main tasks of the jDEECo runtime framework are providing access to the knowledge repository, storing the knowledge of all the running components, scheduling execution of component processes (either periodically or when a triggering condition is met), and evaluating membership of the running ensembles and, in the positive case, carrying out the associated knowledge exchange (also either periodically or when triggered). In general, the jDEECo runtime framework allows both local and distributed execution; currently, the distribution is achieved on the level of the knowledge repository.

2.8.2 Development Progress

In the last project year, we have been working primarly on adding support for simulation to jDEECo. We have managed to integrate jDEECo runtime with (i) the OMNeT++[1] network simulator and (ii) with the MATSim[2] traffic simulator. Whereas integration with (i) allows to realistically simulate jDEECo applications with respect to network infrastructure behavior, integration with (ii) makes it possible to simulate the mobility of jDEECo deployment nodes.

In particular, OMNeT++ provides detailed models of hardware used in contemporary wired and wireless networks together with implementations of different communication protocols recognized so far as standards. MATSim, on the other hand, comes with an extensive agent-based framework. We leveraged its transport simulation functionality by adding the concept of sensors and actuators in jDEECo. By that, each component is capable of retrieving the current geographical location of the node it is deployed on as well as set its position to the desired one.

The integration of these two tools resulted in the jDEECoSim platform. We plan to further extend jDEECoSim by other tools such as SUMO (Simulation of Urban Mobility). Until now, we have validated the platform on a few examples, stemming mainly from the e-Mobility case study.

2.9 jRESP: Runtime Environment for SCEL Programs

2.9.1 Tool Purpose

jRESP is a runtime environment that provides Java programmers with a framework for developing autonomic and adaptive systems based on the SCEL concepts. SCEL [DFLP11, NFLP13] identifies the linguistic constructs for modelling the control of computation, the interaction among possibly heterogeneous components, and the architecture of systems and ensembles. jRESP provides an API that permits using the SCEL paradigm in Java programs.

In SCEL, some specification aspects, such as the knowledge representation, are not fixed but can be customized depending on the application domain or the taste of the language user. Other mechanisms, for instance the underlying communication infrastructure, are not considered at all and remain abstracted in the operational semantics. For this reason, the entire framework is parametrised with respect to specific implementations of these particular features. To simplify the integration of new features, recurrent patterns are largely used in jRESP.

2.9.2 Development Progress

In the forth year of the project, development of jRESP continued in three directions: integration of FACPL in jRESP; development of a high level programming language based on SCEL; integration with jSAM.

Integration with FACPL In jRESP, like in SCEL, policies can be used to authorise local actions and to regulate the interactions among components. Policies can authorise or not the execution of the action (e.g., according to some contextual information) and, possibly, adapt the agent behaviour by returning additional actions to be executed. jRESP provides a common policy interface that can be implemented to integrate different kinds of policies.

The policy interface is currently implemented by two different classes: the permissive policy class and the policy automaton class. The former is the default policy of each node; it allows any action by directly delegating its execution to the corresponding node. The latter policy implements a

[1] http://www.omnetpp.org
[2] http://www.matsim.org

generic policy automaton Π which triggers policy changes according to the execution of agent actions. In particular, a policy automaton consists of a set of automaton states, each of which identifies the possible policies enforced in the node, and of a reference to the current state, which is used to authorise agent actions with respect to the current policies.

When the policy automaton receives a request for the execution of a given action, first of all an authorisation request representing the action is created. This request identifies the action an agent wants to perform (it provides the action name, its argument, its target and the list of attributes currently published in the node interface). The created authorization request is then evaluated with respect to the current policy state. Request evaluation can trigger an update of the current state of the policy automaton. Indeed, for each state, a sequence of transitions is stored in the automaton. Each transition can declare itself enabled or disabled, enabled transitions further provide the new automaton state.

The integration of FACPL in jRESP relies on a specialization of the policy automaton state which wraps the Java-translated FACPL policies, obtained automatically from the FACPL IDE. The specialized class delegates the authorisation decisions to the appropriate FACPL policies. An authorization response consists of the decision itself – permit or deny – and a set of obligations. These are rendered as a sequence of actions that must be performed just after the completion of the currently requested action. If the decision is to permit an action, the corresponding agent can continue as soon as all the obligations are executed. Instead, if the decision is to deny an action, the requested action cannot be performed and the obligations possibly returned must be executed. After their completion, the action previously forbidden can be requested again.

HL-SCEL and integration with jSAM jRESP permits integrating SCEL programming constructs within Java programs. However, to simplify the development process and to simplify the use of formal tools, it could be useful to have an high level programming language that, by enriching SCEL with standard programming constructs (e.g. control flow constructs, such as `while` or `if-then-else`, structured data types,...), simplifies the programming task. For this reason we have defined HL-SCEL, a SCEL inspired high level programming language thought for simplifying design, development and deployment of autonomous and adaptive system. We have also developed an Eclipse plug-in named SCEL SDK that, by relying on XText[3], automatically generates jRESP code that can be used to simulate and execute the programmed system. SCEL SDK has been also integrated with jSAM (see Section 2.10) to provide a simplified interface for supporting quantitative analysis of HL-SCEL programs based on simulations and statistical model checking.

2.10 jSAM: Java Stochastic Model-Checker

2.10.1 Tool Purpose

jSAM is an Eclipse plugin integrating a set of tools for stochastic analysis of concurrent and distributed systems specified using process algebras. More specifically, jSAM provides tools that can be used for interactively executing specifications and for simulating their stochastic behaviors. Moreover, jSAM integrates a statistical model-checking algorithm [CL10, HYP06, QS10] that permits verifying if a given system satisfies a CSL-like [ASSB00, BKH] formula.

jSAM does not rely on a single specification language, but provides a set of basic classes that can be extended in order to integrate any process algebra.

[3] `http://www.eclipse.org/Xtext/`

2.10.2 Development Progress

In the last year of the project, the work around jSAM proceeded towards two directions. On one hand we have integrated new model checking techniques, specifically designed for supporting analysis of large scale systems. On the other hand we have integrated the jRESP simulation environment in order to enable the analysis of SCEL programs via jSAM.

On-the-fly model checking. Model checking approaches can be divided into two broad categories: global approaches that determine the set of all states in a model $\mathcal{M}$ that satisfy a temporal logic formula Φ, and local approaches in which, given a state s in $\mathcal{M}$, the procedure determines whether s satisfies Φ. When s is a term of a process language, the model-checking procedure can be executed "on-the-fly", driven by the syntactical structure of s. For certain classes of systems, e.g. those composed of many parallel components, the local approach is preferable because, depending on the specific property, it may be sufficient to generate and inspect only a relatively small part of the state space. In [LLM14] an efficient, on-the-fly, PCTL model checking procedure that is parametric with respect to the semantic interpretation of the language has been proposed. The proposed model checking algorithm has been integrated in jSAM together with a new module for supporting specification and analysis of systems via the PRISM language.

FlyFast model checker. Typical self-organising collective systems consist of a large number of interacting objects that coordinate their activities in a decentralised and often implicit way. Design of such systems is challenging and requires suitable, scalable analysis tools to check properties of proposed system designs before they are put into operation. The exploitation of mean field approximation in model-checking techniques seems a promising approach to overcome scalability issues raised by the size of such collective systems. In [LLM13a, LLM13b] we have presented a novel scalable, on-the-fly model-checking procedure to verify bounded PCTL properties of selected individuals in the context of very large systems of independent interacting objects. The proposed procedure combines on-the-fly model checking techniques with deterministic mean-field approximation in discrete time. A prototype implementation of the model-checker, named FlyFast, has been integrated into jSAM and used to verify properties of a selection of simple and more elaborate case studies.

SCEL SDK and HL-SCEL. To support design, analysis and deployment of autonomous and adaptive systems developed in SCEL, we have integrated in jSAM a plug-in that, by relying on jRESP simulation environment, enables the use of (some of) the formal tools available in our framework. The proposed plug-in, named SCEL SDK, takes as input HL-SCEL specifications and automatically generates the Java classes used to simulate and execute the considered system.

3 Conclusion

The concluding tool development and integration activities of the ASCENS project have naturally centered around the runtime execution platforms used in the case studies, especially the jDEECo and jRESP frameworks. The updates to these platforms are described in Sections 2.8 and 2.9, respectively. More information about the case studies themselves can be found in the deliverable D7.4.

In general terms, the tool development effort in the fourth year followed the established procedures to assist partner cooperation – in particular, having public tool source repositories where possible, providing tool usage examples, and organizing meetings between tool authors and tool users as necessary.

The same approach – that is, using standard open source development and dissemination practices (source repositories, build tools, update sites) – is part of the effort to enable continuous exploitation

(and possibly further development) of the tools beyond the conclusion of the project. More information on the overall tool landscape is provided in the joint deliverable JD4.2.

References

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time Markov chains. *Transations on Computational Logic*, 1(1):162–170, 2000.

[BBH+12] Lubomir Bulej, Tomas Bures, Vojtech Horky, Jaroslav Keznikl, and Petr Tuma. Performance Awareness in Component Systems: Vision Paper. COMPSAC '12, 2012.

[BCG+12a] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. Adaptable transition systems. In Narciso Martí-Oliet and Miguel Palomino, editors, *WADT*, volume 7841 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2012.

[BCG+12b] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2012.

[BGH+12] Tomas Bures, Ilias Gerostathopoulos, Vojtech Horky, Jaroslav Keznikl, Jan Kofron, Michele Loreti, and Frantisek Plasil. Language Extensions for Implementation-Level Conformance Checking. ASCENS Deliverable D1.5, 2012.

[BKH] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. pages 146–162.

[CL10] Francesco Calzolai and Michele Loreti. Simulation and analysis of distributed systems in klaim. In Dave Clarke and Gul A. Agha, editors, *Coordination Models and Languages, 12th International Conference, COORDINATION 2010, Amsterdam, The Netherlands, June 7-9, 2010. Proceedings*, volume 6116 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2010.

[dA03] Luca de Alfaro. Game models for open systems. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 269–289. Springer, 2003.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC/SIGSOFT FSE 2001*, volume 26(5) of *ACM SIGSOFT Software Engineering Notes*, pages 109–120. ACM, 2001.

[DFLP11] R. De Nicola, G. Ferrari, M. Loreti, and R. Pugliese. Languages primitives for coordination, resource negotiation, and task description. ASCENS Deliverable D1.1, September 2011. `http://rap.dsi.unifi.it/scel/`.

[HG15] Matthias Hölzl and Thomas Gabor. Continuous Collaboration: A Case Study on the Development of an Adaptive Cyber-Physical System. In *Proc. of the International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS), Firenze, Italy*, 2015. to appear.

[HYP06] G. Norman H. Younes, M. Kwiatkowska and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, June 2006.

[KMH14] Annabelle Klarl, Philip Mayer, and Rolf Hennicker. Helena@work: Modeling the science cloud platform. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, volume 8802 of *Lecture Notes in Computer Science*, pages 99–116. Springer Berlin Heidelberg, 2014.

[LLM13a] Diego Latella, Michele Loreti, and Mieke Massink. On-the-fly fast mean-field model-checking. In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2013.

[LLM13b] Diego Latella, Michele Loreti, and Mieke Massink. On-the-fly fast mean-field model-checking: Extended version. *CoRR*, abs/1312.3416, 2013.

[LLM14] Diego Latella, Michele Loreti, and Mieke Massink. On-the-fly probabilistic model checking. In Ivan Lanese, Alberto Lluch-Lafuente, Ana Sokolova, and Hugo Torres Vieira, editors, *Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014.*, volume 166 of *EPTCS*, pages 45–59, 2014.

[MMPT13] Andrea Margheri, Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi. A Formal Software Engineering Approach to Policy-based Access Control. Technical report, DiSIA, Univ. Firenze, 2013. `http://rap.dsi.unifi.it/facpl/research/Facpl-TR.pdf`.

[MZA+12] Lukáš Marek, Yudi Zheng, Danilo Ansaloni, Walter Binder, Zhengwei Qi, and Petr Tuma. DiSL: An extensible language for efficient and comprehensive dynamic program analysis. In *Proc. 7th Workshop on Domain-Specific Aspect Languages*, DSAL '12, pages 27–28, New York, NY, USA, 2012. ACM.

[NFLP13] Rocco Nicola, Gianluigi Ferrari, Michele Loreti, and Rosario Pugliese. A language-based approach to autonomic computing. In Bernhard Beckert, Ferruccio Damiani, FrankS. Boer, and MarcelloM. Bonsangue, editors, *Formal Methods for Components and Objects*, volume 7542 of *Lecture Notes in Computer Science*, pages 25–48. Springer Berlin Heidelberg, 2013.

[Ora12] Oracle. `java.lang.instrument` (Java Platform, Standard Edition 6, API Specification), 2012. `http://docs.oracle.com/javase/6/docs/api/java/lang/instrument/package-summary.html`.

[pth] POSIX Threads. `http://en.wikipedia.org/wiki/POSIX_Threads`.

[QS10] Paola Quaglia and Stefano Schivo. Approximate model checking of stochastic cows. In *Proceedings of the 5th international conference on Trustworthly global computing*, TGC'10, pages 335–347, Berlin, Heidelberg, 2010. Springer-Verlag.

[Zim14] Zimory Software. Zimory Cloud Suite. http://www.zimory.com/, August 2014.