ascens

www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

JD2.2: Modelling and Verification Techniques for SCEs

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: **UJF-VERIMAG**
Author(s): **Jacques Combaz (UJF-Verimag), Mieke Massink (ISTI), Diego Latella (ISTI), Alberto Lluch Lafuente (IMT), Manuele Brambilla, Marco Dorigo, Mauro Birattari (ULB), Mirco Tribastone (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

A central issue of any modeling process is finding the right representation, and in particular the right level of abstraction, in order to study the properties of interest of the target system in an efficient and faithful manner. The inherent complexity of Service Component Ensembles (SCEs) considered in ASCENS makes even more crucial the development of adequate modelling techniques and tools. Proposed models must be able to capture essential aspects of the behavior of SCEs (e.g. interactions between Service Components (SCs), adpative behavior, uncertain or changing environments), and the corresponding analysis techniques must exploit the specific structure of SCEs (e.g. large replication of the same component, hierarchical compositions) so as to discover their properties. This document presents three types of techniques illustrated by their application to the case studies of ASCENS. *(1) Simulation* is in general interesting as it allows to directly include some parts of the real system. We show how it can scale up to vey large number of components and how it can efficiently be used for engineering feedback loops needed in self-adaptation mechanisms. *(2) Quantitative* analysis techniques such as statistical or probabilistic model checking provide abstraction mechanisms adequate for SCEs, as shown for the robotic case study. *(3) Compositional verification* takes advantage of the structure of SCEs allowing incremental verification of large systems.

Contents

1 Integrated Approach for Modelling, Validating and Verifying Ensembles 5

2 The ASCENS Design Flow using BIP 6
2.1 The BIP Framework . . . 7
2.2 Integration of BIP in ASCENS: From SCEL to BIP . . . 8

3 Validation Techniques for Ensembles 9
3.1 Simulation in ARGoS . . . 9
3.2 Simulation of Feedback Loops for Self-Aware and Self-Adaptive Systems . . . 11
3.2.1 Notion of Feedback Loops . . . 12
3.2.2 Key Goals of the Plug-in . . . 12
3.2.3 Application to the E-Mobility System . . . 13
3.3 Soft-Constraint Based Optimization . . . 14
3.4 Quantitative Analysis . . . 16
3.4.1 Statistical and Probabilistic Model Checking . . . 16
3.4.2 Fluid-Flow Analysis . . . 18

4 Compositional and Incremental Verification Techniques 19

5 Application of the ASCENS Design Flow to the Robotics Case Study 21
5.1 Garbage-Collecting Robots . . . 21
5.2 Requirements Specification with SOTA . . . 22
5.3 SCEL model of the case study . . . 23
5.4 Quantitative Analysis and Validation . . . 24
5.5 Quantitative Verification of SOTA Requirements . . . 24

6 Conclusion 25

A Non-Ascens Tools used for Verification 26
A.1 PEPA . . . 26
A.2 Bio-PEPA . . . 26
A.3 CIAO Language . . . 27

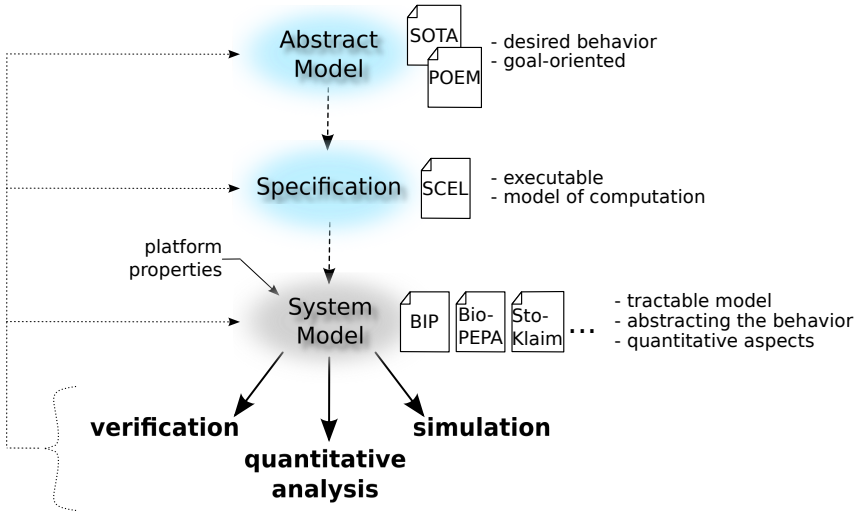


Figure 1: The ASCENS Design Flow.

1 Integrated Approach for Modelling, Validating and Verifying Ensembles

System design differs radically from pure software design in that it must account not only for functional requirements but also for extra-functional requirements regarding the use of execution platform resources, such as time, memory, and energy. Meeting extra-functional requirements requires evaluation of how design choices affect overall system behaviour. It also implies a deep understanding of how components of the system are interrelated, and how they interacts with the underlying execution platform. Yet system designers currently lack rigorous techniques for deriving global models of a given system from software specifications and execution platform. We define a rigorous design flow as one that guarantees essential system properties. Figure 1 illustrates the rigorous system design flow proposed for ASCENS. It involves the following distinct steps.

1. Express the requirements in terms of an abstract model representing the desired behavior for the system, using frameworks such as SOTA or POEM (see Deliverable D2.1).

2. Build a specification provided as a SCEL program which is executable by means of Java code direclty associated to the primitives of SCEL. It can be used for early validation (e.g. simulation) of the application, taking only functionality into account without considering any execution platform.

3. Refine the previous specification into a system model by integrating architectural constraints such as properties of the hardware target platform and the mapping of the components on the platform. For static architectures (i.e. statically known SCEs), BIP system models can be automatically obtained from SCEL specifications as explained in Section 2.2. This allows to use the whole BIP toolbox including tools for simulation, statistical model checking, and compositional verification (see Section 2.1). System models can also be derived manually for specific validation techniques and tools, e.g. expressed as Bio-PEPA models for fluid-flow analysis (see Section 3.4.2).

4. Validate the system model using simulation, quantitative analysis, and/or verification tech-

niques. If this step is successful the system is implemented and tested in real-life conditions. Otherwise either the abstract model or the SCEL specification must be modified, of the system model must be refined.

One of the main challenges of this process is finding the right system models in order to be able to perform the validation step in an efficient and correct manner. The inherent complexity of Service Component Ensembles (SCEs), and in particular the number of components they involve, can easily lead to non-tractable models. This deliverable focuses on the models and analysis techniques proposed for ASCENS and applicable to SCEs. We mainly consider three different types of techniques which are presented in conjunction with their corresponding models. We will also explain how these models relate to SCEL specifications from which they should be derived.

(1) Simulation is always interesting in system design since it allows to directly include some parts of the real system in order to validate them. However, even simulation is often not efficient enough to be usable for large systems. Section 3.1 presents ARGoS, a simulator that can be used for early validation of swarm robotic applications, allowing to speed-up their design with respect to the standard practice which is based on physics-based simulation. We also proprose in Section 3.2 a simulation framework adapted to the engineering of feedback loops needed by self-adaptation mechanisms.

(2) Quantitative analysis techniques (Section 3.4) such as statistical or probabilistic model checking and fluid-flow analysis can efficiently answer to performance questions thanks to appropriate abstraction mechanisms. Statistical Model Checking (SMC) approaches can be seen as an improvement of purely simulation-based techniques since they guarantee results with respect to a user-defined level of confidence. This is obtained in a SMC model checker by computing a number of executing sequences sufficient to get the coverage of the system behavior required by the level of confidence.

We also propose in Section 4 a *compositional verification (3)* method that permits incremental verification of large systems by taking advantage of their structure. It has been implemented in the BIP framework which is presented in Section 2.1. The BIP framework can be considered as a possible instanciation of the ASCENS design flow as it includes a large diversity of validation tools including simulation tools, satistical model checking tools, and verification tools. The connection with the rest of the ASCENS design flow is ensured by the translation of the SCEL language into BIP which is described further.

Section 3.3 present the soft constraint programming model that is suitable for expressing multi-criteria optimization problems. We show how it can be applied to solve the e-mobility optimization problem consisting in finding a journey for a car that optimizes both time and energy consumption. Solutions are computed using CIAO Prolog given a model of the road network and associated time and energy costs.

The rest of the deliverable illustrates the proposed design flow by its application to the robotics case study of ASCENS (see Section 5), and provides a conclusion and future work.

2 The ASCENS Design Flow using BIP

In this section we give an overview of the BIP methodology and associated tools. The BIP framework can be integrated in the ASCENS design flow using translation of SCEL specifications into BIP models, which is presented in Section 2.2. The BIP framework includes a large diversity of existing validation techniques which have been improved during the first two years of the project (see Deliverables D5.1 and D5.2). They include simulation, satistical model checking, and compositional verification presented in in Section 4.

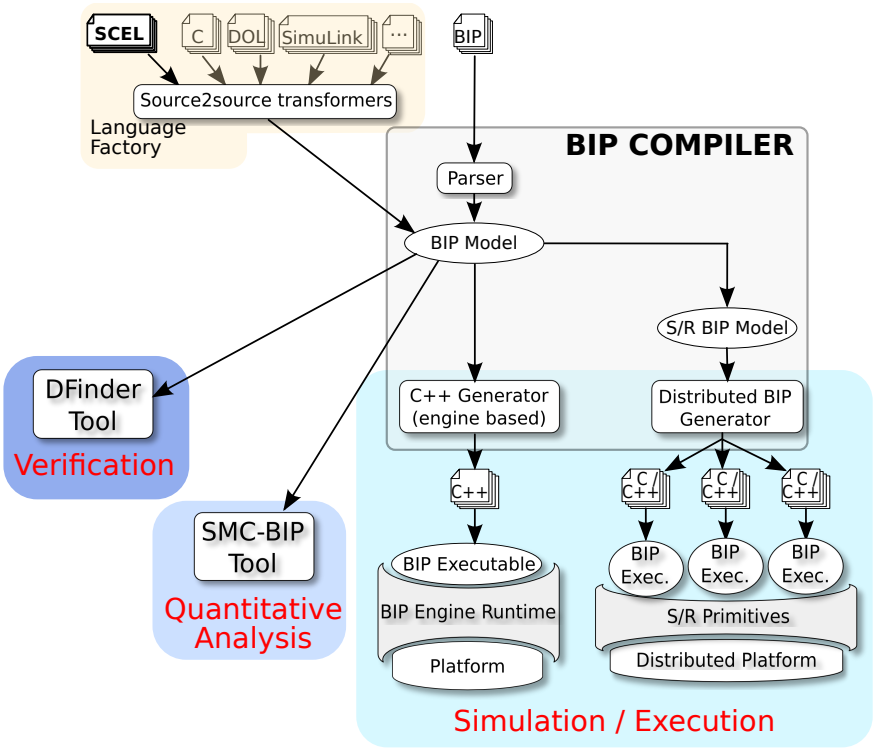


Figure 2: Overview of the BIP tool-chain for ASCENS.

2.1 The BIP Framework

The behaviour, interaction, priority (BIP) component framework was developed to support rigorous system design, that is, guaranteeing essential system properties. The BIP framework is

- model-based, describing all software and systems according to a single semantic model. This maintains the flow's overall coherency by guaranteeing that a description at step $n + 1$ meets essential properties of a description at step n.
- component-based, providing a family of operators for building composite components from simpler components. This overcomes the poor expressiveness of theoretical frameworks based on a single operator, such as the product of automata or a function call.
- tractable, guaranteeing correctness by construction and thereby avoiding monolithic a posteriori verification as much as possible.

BIP supports the construction of composite, hierarchically structured components from atomic components characterised by their behaviour and interfaces. It lets developers compose components by layered application of interactions and priorities. This enables an expressiveness unmatched by any other existing formalism. Architecture is a first-class concept in BIP, with well-defined semantics that system designers can analyse and transform.

BIP differs significantly from existing component frameworks for software engineering. These often use multi-threaded programming and point-to-point interaction mechanisms, such as function calls, for coordination between components. In contrast, BIP executes atomic components concurrently and coordinates them in terms of high-level mechanisms such as protocols and scheduling

policies. Because BIP focuses on the organisation of computation between components, it can be viewed as an architecture description language (ADL). Like other existing ADLs, such as Acme (www.cs.cmu.edu/ acme, [GMW97]) and Darwin [MK96] BIP uses the connector concept to express coordination between components. Nonetheless, connectors in BIP are stateless. The architecture, consisting of connectors and priorities, is clearly distinguished from behaviour. Another significant difference from other frameworks is that BIP is intended for system modelling. It directly encompasses timing and resource management. Other system modelling formalisms either seek generality to the detriment of rigorousness, such as (Systems Modelling Language (SysML) and (Architecture Analysis and Design Language (AADL; `http://standards.sae.org/as5506a`) or limit their scope to specific computation models, such as Ptolemy [EJL+03]. We have lately considered an extension of BIP called DyBIP which encompasses dynamic expressions of architectures [BJMS12], which could be very useful in the context of ASCENS where interactions between components depend on the dynamics of the system, e.g. in the robotic case study a robot can only interact with its neighbourhood.

The BIP framework is supported by a tool-chain implementing our design flow, including model-to-model transformations [BJS10] and code generators (see Figure 2). We target centralised platforms for which we provide an engine-based execution [BBBS08] of the generated code, as well as distributed platform that requires transformations to send/receive BIP models [GR12] that can be mapped directly on a distributed platform. We also developed validation and verification tools of BIP models. D-Finder is a verification tool for deadlock freedom properties implementing compositional and incremental verification techniques. It has been successfully applied to large sets of interacting components [BBL+10, BBB+11], and thus seems to be suited to ASCENS applications. We also developed validation tool SMC-BIP implementing statistical model-checking (SMC) of BIP models. Given a stochastic BIP model, SMC-BIP check for properties expressed as P-LTL expressions, based on a partial coverage of the state space of the model, that is, by running a suite of simulation runs. Parameters representing confidence are used to set set the acceptable probability for the tool to give a wrong answer. From these parameters, SMC-BIP determines automatically the number of simulations to be run in order to guarantee the considered level confidence. Statistical model checking can scale to large systems using stochastic abstractions [BBB+12], and is particularly adapted for checking non-functional properties.

2.2 Integration of BIP in ASCENS: From SCEL to BIP

The first step when using the BIP framework consists of generating a BIP model for the application software. We have developed a general method for generating BIP models from languages with well-defined operational semantics. It involves the following steps for a given application software written in a language $\mathcal{L}$.

1. Translating the source language $\mathcal{L}$'s atomic components into BIP components. The translation focuses on the definition of adequate interfaces. It encapsulates and reuses the application software's data structures and functions.

2. Translating the coordination mechanisms between application software components into the target BIP model's connectors and priorities.

3. Generating a BIP component that models $\mathcal{L}$'s operational semantics. This component plays the role of an engine coordinating the execution of the application software components.

We developed BIP model generators for several programming models used by embedded system developers (the source-to-source transformers in Figure 2). The generated models preserve the structure

of the initial programs, their size is linear with respect to the initial program size, and they are easy for the system developers to understand.

To integrate the BIP framework into the ASCENS design flow, we instanciate this general translation method for the SCEL language[1]. We considered a subset of SCEL allowing the generation of static BIP architectures, since most of BIP tools do not handle the dynamic extensions of BIP presented in Deliverables D2.1 and D2.2. It includes to the following restrictions.

- process policies are assumed to be monitors expresses as Petri Net with priorities
- process variables cannot be used in patterns
- recursive creation of new locations is forbidden

The different elements of this subset of SCEL are translated into BIP as explained follows.

Ensembles are translated into BIP composite components (i.e. hierarchical components) containing the translation of the respective knowledge and of the enclosed ensemble. Interactions are obtained by combining interactions from the knowledge and the contained ensemble and by adding all interactions from the ensemble not involving the local knowledge. Restriction of locations names is rendered in BIP by making private (not exported) any interaction involving the bound location name.

Components are translated into BIP composite components composed of:

- an atomic component representing the translation of the local knowledge
- a component corresponding to the process
- components derived from process definitions (which are supposed to be local to any node).

Atomic components for the knowledge export four ports: three of them corresponds to *put*, *get*, and *read* operations, and fourth port is used for receiving values after performing *put* and *get* operations. Process definitions are translated into BIP components where the behaviour corresponds to a PN with an initial transition corresponding to definition invocation. The rest of the net corresponds to the PN of the body of the definition. At the end of the execution of the body a token has to be added to the initial place in order to make available a new copy of the definition. It is possible to model the fact that multiple copies of same definition can run concurrently only the maximal number of these copies can be bounded statically.

3 Validation Techniques for Ensembles

3.1 Simulation in ARGoS

The aim of the robotics case study is to apply the methodologies developed in the foundational workpackages to the problem of designing control strategies for swarms of robots.

Traditionally, control strategies for swarms of robots are prototyped in simulation, and subsequently tested in targeted real-world experiments. The workflow is composed of two main cycles as shown in Figure 3. Initially, the designer sketches an idea and implements it in simulation. The designer conducts test experiments in simulation to assess the performance of the idea. If performance is not satisfactory, the designer refines the idea and its implementation. This first work cycle terminates when satisfactory performance is achieved. In this cycle, the use of physics-based simulations is common practice in robotics because the obtained results usually display a high degree of correspondence with real-world experiments. Once satisfactory performance is achieved in simulation, the

[1] Deliverable D1.1 provides a full description of SCEL.

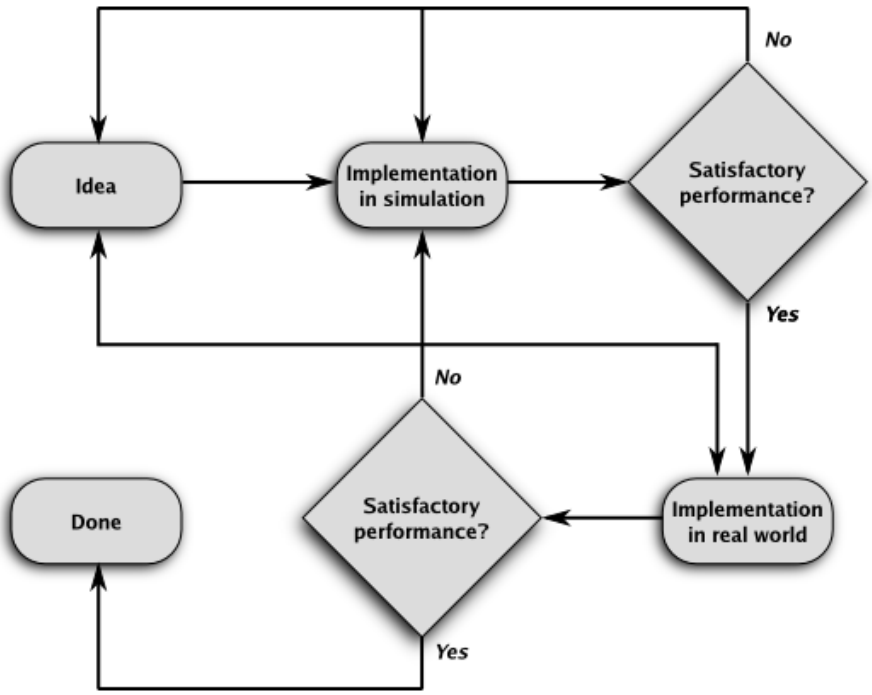


Figure 3: Traditional design workflow in swarm robotics.

idea is ported to robots for real-world experimentation. The refinement of the idea can entail either a correction of the real-world implementation, or even a return to simulation. This latter case typically occurs when the simulation is not accurate enough to capture the relevant aspects of the experiment. The end product of this workflow is a system that works both in simulated and real-world experiments.

The main defects of the traditional work cycle lie in the fact that the quality of the final result, as well as the time necessary to obtain such result, are completely dependent in the ingenuity and skill of the designer. The methodologies and tools studied in the ASCENS project aim to provide semi-automatic methods that ease design and render the quality of the final result less dependent on the designer.

The use of ASCENS tools affects various parts of the traditional robotics design workflow, as depicted in Figure 4. The SCEL language and its analysis tools, for instance, aim to allow the designer to refine the idea even before its implementation in physics-based simulations. In particular, SCEL allows the designer to reason on the idea abstracting away those details that, in this first design stage, would constitute only an unnecessary complication. In addition, the idea can undergo early validation tests through systems such as BIP. The net result of this additional work cycle is an abstract prototype already amended of issues whose discovery, traditionally, would have required extensive testing in physics-based simulations. Furthermore, in the ASCENS design workflow, the results obtained through physics-based simulations can be used both to refine the idea (as in the traditional workcycle) and to refine its SCEL model, enabling further abstract analysis. The same can be said for the results obtained in real-world experiments.

The application of the ASCENS design workflow on challenging swarm robotics problems is a mid- to long-term perspective. In the short term, however, this design workflow finds its usefulness in the validation of the ASCENS tools and methods against well-known solutions in the literature. In this perspective, the ASCENS project provides an important, mostly unexplored occasion to link practical robotics design to theoretical approaches.

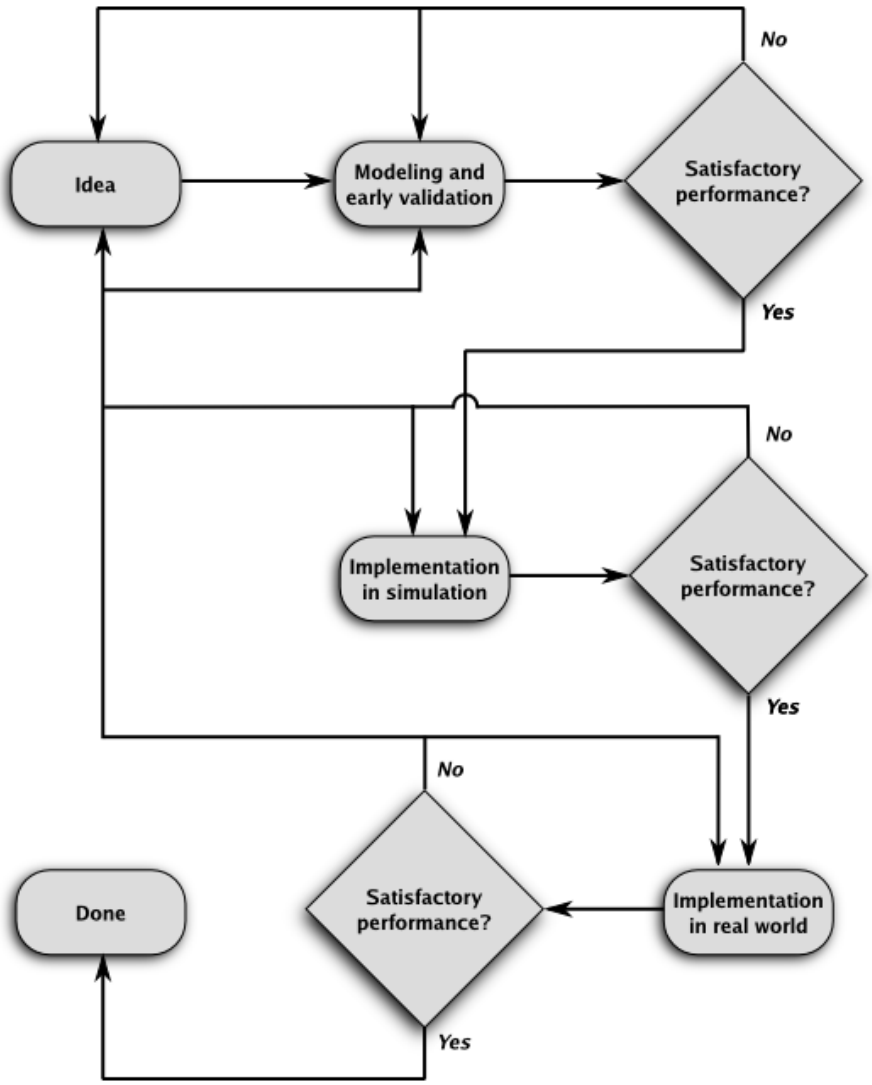


Figure 4: Design workflow in swarm robotics using ASCENS tools.

In the development of the ASCENS design workflow, the ARGoS robot simulator [PTO+11] will be a fundamental tool. ARGoS is capable to simulating ensembles composed of thousands of robots very efficiently. Also, ARGoS allows the designer not only to execute highly complex robotics simulations, but also to seamlessly port code from simulation to real robots. In other words, using ARGoS, we can collect data both in simulated and in real-world experiments in a consistent environment, thus making comparisons among different approaches (to model the robots, to control them, etc.) easy and meaningful.

3.2 Simulation of Feedback Loops for Self-Aware and Self-Adaptive Systems

The increasing complexity and dynamics of the environment in which software systems are deployed call for solutions to make such systems *autonomic*, i.e., capable of dynamically self-adapting their behavior in response to changing situations [**?**]. SOTA (state of the affairs) [**?**] introduces a general model, for modeling the self-awareness and self-adaptation requirements of adaptive systems. In the SOTA space, a system is *self-aware* if it can autonomously recognize its current position and direction

of movement in the space. *Self-adaptation* implies that the system is able to dynamically direct its trajectory in the SOTA space. Such capability necessarily requires the existence of *feedback loops* inside the system, to detect its current trajectory, and properly correct it on need to reach specific regions of the space, corresponding to specific application goals. To achieve this, SOTA defines several *architectural design patterns* [**?**] in which feedback loops are organized.

Many approaches on software engineering of self-adaptive systems propose solutions for their development, analysis and validation methods. However, few approaches (e.g., [**?**, **?**, **?**]) provide explicit support for the engineering of feedback loops. In particular, limited attention has been given to providing tool support for simulating these feedback loops, and for understanding how such feedback loops should be architected. Thus, as a specific contribution, we are currently developing an *Eclipse-based simulation plug-in* [**?**] to support the engineering (i.e., modeling, animating and validating) of self-adaptive systems based on feedback loops. The plug-in is developed using the IBM Rational Software Architect Simulation Toolkit 8.0.4. Our approach is explored and validated using the basic scenario (S_0) of the e-mobility case study [**?**].

The rest of this section is organized as follows. In Section 3.2.1, we outline the notion of feedback loops explored in SOTA to express self-awareness and self-adaptation. The key goals of our plug-in are presented in Section 3.2.2. An example of the simulation is provided in Section 3.2.3.

3.2.1 Notion of Feedback Loops

A *feedback loop* is the part of the system that allows self-adaptation towards goals' achievement, i.e., self-adjusting behavior in response to system changes (or environmental changes) [**?**]. It provides a generic mechanism for adaptation where it provides means for inspecting and analyzing the system at the SC or SCE level and for reacting accordingly.

Realistically, a feedback loop needs to contain multiple control loops forming a feedback structure, which allows it to work independently and in coordination [**?**]. These multiple control loops can interact using two basic mechanisms of loop interaction: *hierarchy* and *stigmergy* [**?**]. They can coordinate and support adaptation using two basic mechanisms: *inter-loop* and *intra-loop* coordinations [**?**]. These loops extend the IBM's MAPE-K (i.e., monitor, action, plan, execute and knowledge) [**?**] model of adaptation. Also, two types of feedback loops can be identified depending on the nature of feedback, i.e., *positive* and *negative* [**?**].

We use UML 2 activity diagrams as the primary notation to model the behavior of feedback loops [**?**]. In a feedback loop, all the actions are not necessarily performed sequentially. An iterative process allows to revise certain decisions if required, and therefore, activity diagrams are effective to design the feedback loops. Activity partitions are used to represent the SCs.

3.2.2 Key Goals of the Plug-in

As feedback loops can be organized using many architectural patterns, it is important to provide tools to support their engineering. The key goals of our plug-in include [**?**]:

- Modeling SOTA patterns using UML 2 where the patterns' structural and behavioral information modeled using activity, sequence and composite structure diagrams.
- Visually animating SOTA patterns' behavior during execution to expose runtime view of the simulated model (next element to execute, executed element, active states, tokens).
- Animating composite structures of SOTA patterns, e.g., interaction messages and token flows, and execution history information of the simulation.

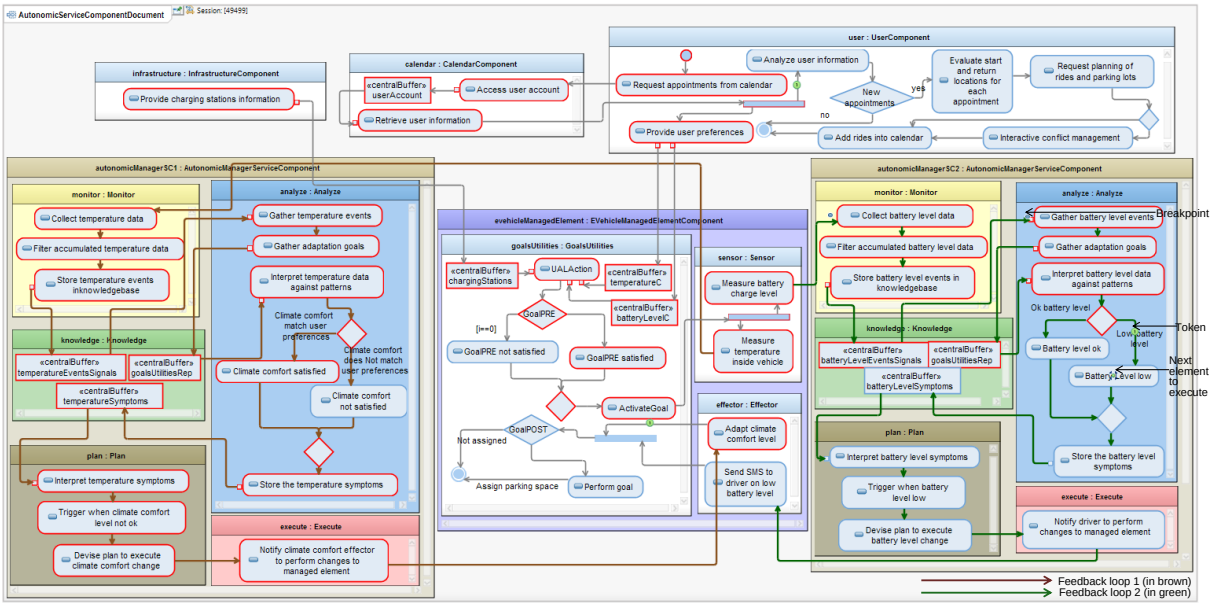


Figure 5: Plug-in: SOTA Autonomic SC pattern simulated for an e-mobility scenario [**?**].

- Model-level debugging and control of patterns execution. This adds breakpoints and other commands, e.g., stepping, suspend, resume, terminate.
- Simulating event-driven models of the patterns, which can occur due to the execution of a model element or the engineer manually sending an required event.
- Run-time prompting during patterns simulation, which allows the engineer to interact on how to proceed with the execution of an informal model construct.

In order to clarify the ideas behind the plug-in, we apply two key SOTA patterns at the SC and SCE levels, i.e., the *Autonomic SC pattern* and the *Centralized Autonomic SCE pattern*. The *Autonomic SC pattern* is characterized by the presence of an explicit, external feedback loop to direct the behavior of the managed element [**?**]. An autonomic manager (AM) handles adaptation of the managed element. Several AMs can be associated to the managed element, each closing a feedback loop devoted to control a specific adaptation aspect of the system. The *Centralized Autonomic SCE pattern* has a centralized global feedback loop, which manages an higher-level adaptation of behavior of multiple autonomic components. The adaptation is handled by a super AM, and like an AM it has the IBM's MAPE-K adaptation model. This is while the single SCs are able to self-adapt their individual behavior using their own external feedback loops.

3.2.3 Application to the E-Mobility System

The basic scenario (S_0) of the e-mobility case study is used to explore and validate our simulation. Let us consider a situation where a user intends to travel for an appointment or meeting at a particular destination. First, the user drives the electric vehicle to the car park, then parks the car and walks to the meeting location. During walking time and meeting time the electric vehicle can be recharged.

In this example, the main SCs are the user, the electric vehicle, the parking lots and charging stations. These SCs can be conceptually modeled as SOTA entities moving in the SOTA space. Each of these can be modeled in terms of entities having goals, and utilities (at individual or global level) [**?**] that describe how such goals can be achieved. The main SCEs are (i) the temporal orchestration of the

user, the electric vehicle, a parking lot and a charging station (assigned infrastructure), (ii) a collection of available parking lots and (iii) charging stations. Each SC and SCE of this mobility scenario can be described using (i) the SOTA goals and utilities, (ii) the awareness being monitored for the managed element, (iii) any contingencies that can occur, and (iv) the corresponding self-adaptive actions using feedback loops.

For example, let us consider the electric vehicle SC, which is the central SC within the mobility scenario. It interfaces with both the user and the infrastructure SCs during driving, and with the infrastructure SCs only during parking or walking. The user SC provides travel input to the electric vehicle. The goal of the electric vehicle is to reach the destination with the planned energy and at the planned time. An utility can be that the battery's state of charge should not reach 'low' until the electric vehicle reaches its destination which is its goal. The awareness being monitored are the battery state of charge, the vehicle's current location, and time. Contingency situations that require self-adaptive behavior are (i) the unavailability of a parking lot, (ii) the electric vehicle cannot reach the destination on time with the planned energy, and (iii) the user overrides the plan. Possible self-adaptive actions are (i) change the booking, (ii) change the route, and (iii) change the driving style.

In order to handle adaptation of a managed element, we provide separate AMs for each SOTA awareness dimension. The electric vehicle SC has three AMs defined to handle adaptation on battery state of charge, climate comfort requirements of the user, and location. Any self-adaptive behavior on routing needs to be handled at the SCE level as these actions are applicable to both the user and the electric vehicle SCs. Thus, a super AM has been defined to handle adaptation on routing. Fig.5 provides an example of the plug-in in operation for the electric vehicle SC with two AMs to handle adaptation on battery state of charge and climate comfort requirements (see [**?**] for details).

Engineering e-mobility as a decentralized system of autonomous (self-aware and self-adaptive) SCs and SCEs is a rather challenging task for software architects. The complexity is associated to the number of SCs, the SCE orchestration, AMs and super AMs closing multiple, interacting feedback loops. It has been the aim of our work to provide engineering support, i.e., modeling, animation and validation, to the software architect in order to easily grasp the complex setup.

3.3 Soft-Constraint Based Optimization

Classical constraint satisfaction problems (CSPs) [Tsa93] represent an expressive and natural formalism useful to specify different types of real-life problems. A CSP can be described as a set of variables associated with a domain of values, and a set of constraints. A constraint is just a limitation of the possible combinations of the values of some variables. Therefore, solving a CSP consists in finding an assignment of values to all its variables guaranteeing that all constraints are satisfied.

The *soft* framework [BMR95] extends classical constraints by adding to the usual notion of CSP the concept of a structure representing the levels of satisfiability or the costs of a constraint. Such a structure is represented by a semiring, that is, a set with two operations: one ($+$) is used to generate an ordering over the levels, while the other one ($\times$) is used to define how two levels can be combined.

Constraint logic programming (CLP) [JL87] extends logic programming (LP) by embedding constraints in it: term equalities is replaced with constraints and the basic operation of LP languages, i.e. the unification, is replaced by constraint handling in a constraint system. It therefore inherits the declarative approach of LP, according to which the programmer specifies what to compute while disregarding how to compute it, by also offering efficient constraint-solving algorithms.

However, only classical constraints can be handled in the CLP framework. So, in [BMR01], it has been extended to also handle soft constraints. This has led to a high-level and flexible declarative programming formalism, the *Soft CLP* (SCLP), allowing us to easily model and solve real-life problems involving constraints of different types. Roughly speaking, SCLP programs are logic programs where

constraints are represented by predicates defined by clauses whose body is a value of the semiring modelling the costs. The flexibility is due to the fact that the same framework can be used to handle different kinds of soft constraints by simply choosing different semirings. It can indeed be used to handle fuzzy, probabilistic, prioritized and optimization problems, as well as classical constraints.

In [MM12], we propose the SCLP framework as a high-level specification formalism useful to naturally model and also solve the e-mobility optimization problem [HZWS12].

As an example, here we show how we can model the optimization sub-problem consisting in finding the best trips in terms of travel time and energy consumption. This problem substantially coincides with the multicriteria version of the shortest path problem [BMRS10]. It is then used to model the e-mobility optimization problem consisting in finding the optimal journey. In order to actually execute the SCLP program, we propose CIAO Prolog [BCC$^+$97], a system supporting constraint logic programming (see Appendix A.3), by explicitly implementing the soft framework.

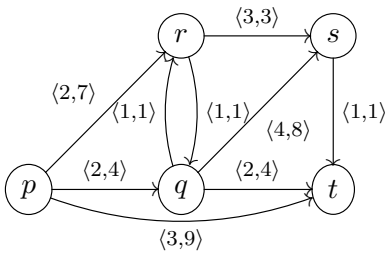
Modelling the trip optimization problem. The road network is represented by a directed graph $G := (N, E)$, where each arc $e \in E$ from a node p to a node q has a label $\langle c_T, c_E \rangle$, i.e., a pair whose elements represent the costs, respectively in terms of time and energy consumption, of the arc.

Given the road network G, such as the one on the top of Fig. 6, a source node n_s and a destination node n_d, the problem consists in finding all the best paths between n_s and n_d in terms of time and energy consumption. Note that, since the costs of the arcs are elements of a partially ordered set, the solution can contain several paths, that is, all paths which are not dominated by others, but which have different incomparable costs. For example, if we want to know the best paths from p to t in the graph of Fig. 6, the solution will contain both the paths $\{p, t\}$ with cost $\langle 3, 9 \rangle$ and $\{p, q, t\}$ with cost $\langle 4, 8 \rangle$. The former is indeed better in terms of time while the latter in terms of energy consumption.

The CIAO program modelling the problem is shown in Fig. 6. We have a set of clauses modelling the road network, that is, a set of facts modelling all the edges of the graph. Each fact has the shape $edge(n_s, n_d, [c_T, c_E])$, where n_s represents the source node, n_d represents the destination node and the pair $[c_T, c_E]$ represents the costs of the edge in term of time and energy. Note that, differently from what would happen in the pure SCLP framework, these facts (representing constraints) have the cost in the head of the clauses and not in the body. This is needed for implementing the soft framework.

Moreover, there are two clauses $path$ describing the structure of paths: the upper one models the base case, where a path is simply an edge, while the lower one represents the recursive case, where a path is an edge plus another path. The head of the path clauses has the following shape $path(n_s, n_d, L_N, L_V, [c_T, c_E], Lim)$, where n_s and n_d are respectively the source and destination nodes, L_N is the list needed to remember, at the end, all the visited nodes, L_V is the list of the already visited nodes needed to avoid infinite recursion where there are graph loops, $[c_T, c_E]$ is used to remember the costs of the path, and finally, Lim represents the maximum amount of energy that the vehicle can consume. It is used to retrieve only the paths with a total cost in terms of energy equal to or less than the passed value.

The $times$ and $plus$ clauses model the two operations of the soft framework. In particular, the first clause models the multiplicative operation of the semiring allowing us to compose the global costs of the edges together. The $plus$ predicate instead mimics the additive operation and it is useful to find the best, i.e. non-dominated, paths. The $plus$ predicate is indeed used in the body of the $paths$ clause, which collects all the paths from a given source node to a given destination node and returns the best solutions chosen with the help of the $plus$ predicate. So, if we want to know the best paths, in the graph of Fig. 6, from p to t with a total cost in terms of energy consumption less than or equal to 10, we have to perform the CIAO Prolog query $paths(p, t, 10, BestPaths)$, where the $BestPaths$ variable will be instantiated with the list containing all the non-dominated paths.




```
:-module(paths,_,_).
:-use_module(library(lists)).
:-use_module(library(aggregates)).

minPair([T,E],[T1,E1]):-
  T < T1,
  E < E1.

times([T1,E1],[T2,E2],[T3,E3]):-
  T3 = T1 + T2,
  E3 = E1 + E2.

plus([],L,[]).
plus([[P,T,E]|RestL],L,
  [[P,T,E]|BestPaths]):-
  nondominated([T,E],L),
  plus(RestL,L,BestPaths).
plus([[P,T,E]|RestL],L,BestPaths):-
  \+nondominated([T,E],L),
  plus(RestL,L,BestPaths).

nondominated([T,E],[]).
nondominated([T,E],[[P,T1,E1]|L]):-
  \+minPair([T1,E1],[T,E]),
  nondominated([T,E],L).
```

```
edge(p,q,[2,4]).  edge(q,t,[2,4]).
edge(p,r,[2,7]).  edge(r,s,[3,3]).
edge(p,t,[3,9]).  edge(r,q,[1,1]).
edge(q,r,[1,1]).  edge(s,t,[1,1]).
edge(q,s,[4,8]).

path(X,Y,[X,Y],_,[T,E],Lim):-
  edge(X,Y,[T,E]),
  E =< Lim.

path(X,Y,[X|L],V,[T,E],Lim):-
  edge(X,Z,[T1,E1]),
  nocontainsx(V,Z),
  path(Z,Y,L,[Z|V],[T2,E2],Lim),
  times([T1,E1],[T2,E2],[T,E]),
  E =< Lim.

paths(X,Y,Lim,BestPaths):-
  findall([P,T,E],path(X,Y,P,[X],[T,E],Lim),ResL),
  plus(ResL,ResL,BestPaths).
```

Figure 6: The road network and the CIAO program modelling the trip level optimization problem.

3.4 Quantitative Analysis

Quantitative analysis techniques are used to answer to performance questions for a system. Thanks to appropriate abstraction mechanisms, they can be efficient even for large and complex systems such as SCEs considered in ASCENS. For instance, fluid-flow analysis (see Section 3.4.2) uses continuous-state approximations of Markov chains by means of ordinary differential equations which are insensitive to the number of components, thus completely avoiding state-space explosion problems rised by large scale systems. Statistical Model Checking (SMC) (see Section 3.4.1) approaches can be seen as an improvement of purely simulation-based techniques since they guarantee results with respect to a user-defined level of confidence. This is obtained in a SMC model checker by computing a number of executing sequences sufficient to get the coverage of the system behavior required by the level of confidence.

3.4.1 Statistical and Probabilistic Model Checking

Statistical Model-checking of Robot Swarm Decision-making

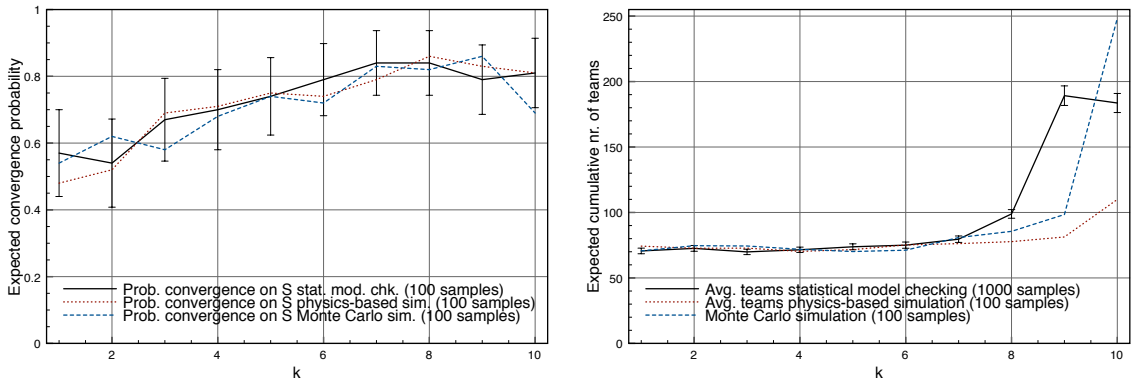
Model checking has first been developed in a non-quantitative setting. Pioneers of this technique, starting their developments in the early eighties of the previous century, are, among others, Edmund Clark, Allen Emerson, Joseph Sifakis (see e.g. [CES09]) and Gerard Holzmann (see e.g. [Hol91]). This verification technology provides algorithmic means to determine whether an abstract model of,

for example, a hardware of software design, satisfies a formal property expressed as a temporal logic formula. More recently, model checking techniques have been extended to deal with quantitative performance aspects of concurrent systems and related probabilistic versions of the temporal logics for the specification of quantitative properties have been formulated, such as the Continuous Stochastic Logic (see e.g. [ASSB00, BKH99]). Efficient model checking methods for these stochastic variants are based on well-known numerical algorithms for the calculation of transient and steady state probabilities of continuous and discrete time Markov chains. Although *stochastic model checking*, as this probabilistic variant is called, may generate very accurate answers, it relies on building the state space of the complete underlying Markov chain of the abstract system model, which (currently) restricts its realistic applicability to system models of at most 10^7 states.

A recently proposed related way to analyse large concurrent systems is via *statistical* model checking. In its most general form, statistical model checking is an analysis method in which a logical formula, formalising a probabilistic property of interest, is automatically checked against a set of randomly generated simulation runs of a high-level model of the system. The probability that the formula holds for the model is then estimated via *statistical analysis* rather than numerical analysis. This has various consequences. On the one hand, statistical model checking can deal with system models that have very large state spaces because only a set of paths need to be generated instead of the whole state space. On the other hand, in cases in which high accuracy is required the set of paths that need to be generated may be huge as well. So, in case of very large systems and when high accuracy is not the main issue, statistical model checking may be the right option. Various statistical techniques have been implemented and added to existing stochastic model checkers such as PRISM (see e.g. [KNP11]). Among these are techniques to approximate the probability with which a formula holds and techniques to establish whether such a probability is above or below a certain given bound. The former is based on various confidence interval methods, whereas the latter is based on hypothesis testing, in particular Wald's sequential probability ratio test (see e.g. [YKNP06]).

In Massink et al. [MBL$^+$12] robot swarm decision-making has been modelled in Bio-PEPA [CH09] and analysed using statistical model checking and fluid-flow analysis (using the Bio-PEPA tool suite [CDG$^+$09] and PRISM). In particular confidence interval (CI) methods were used to estimate probability and rewards. Reward structures can be used, for example, to count the number of times that certain actions occur. The validity of Bio-PEPA and statistical model checking in the field of swarm robotics is illustrated by modelling collective decision-making in a swarm robotics system, a strategy that was originally introduced and analysed in [MFS$^+$11], and by comparing the result of different analyses with those available in the literature [MFS$^+$11]. This strategy consists in a robot swarm residing in a start area and that has the task to discover the shortest of two paths to a goal area. The robots do so by remembering the last path they have taken when returning from the goal area to the start area. When leaving the start area, they form small teams, i.e. simple ensembles, of three randomly selected robots from the start area that decide which path to take as a team by means of a majority vote. This strategy is a variant of the well-known double-bridge experiment known from ant colony behaviour [GADP89].

Fig. 7(a) shows the probability that the system converges to the short path for different numbers of active teams k in the system (k ranging from 1 to 10 for a total swarm size of 32 robots). The convergence property has been formalised as a CSL (Computation tree Stochastic Logic [ASSB00]) reachability property. Fig. 7(b) shows an example of a reward formula, in particular the expected number of team formations until convergence on one of the paths occurs. Both analyses were based on a maximal path length of 20,000, a confidence level $\alpha = 0.01$ and 100 (resp. 1000) sample paths. The width of the calculated confidence intervals are shown as error bars. The results obtained with stochastic model checking are presented together with the results from [MFS$^+$11] that where obtained by Monte Carlo simulation and Physics based simulation of the same system. Overall, a good corre-



(a) Probability of convergence on the short path (100 samples)

(b) Expected number of team formations until convergence (1000 samples)

Figure 7: Stochastic Model Checking results for swarm robotics.

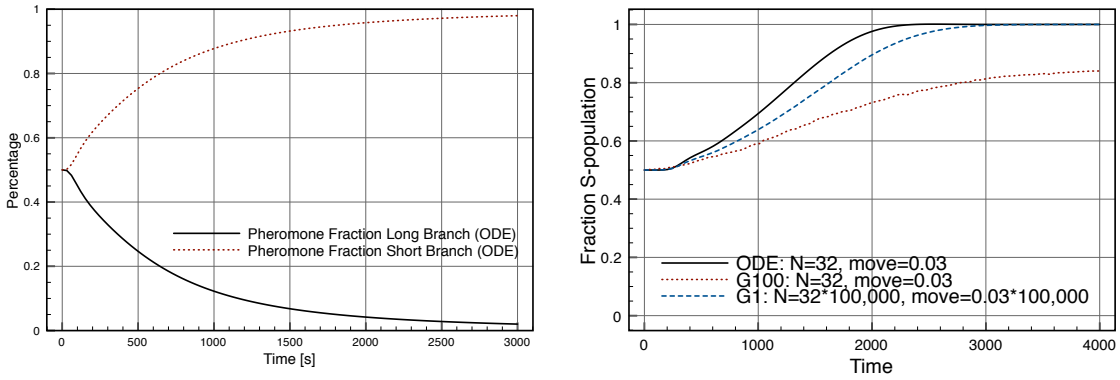
spondence can be observed between the various approaches. The research has been jointly carried out in a collaboration of researchers with different, complementary expertise, from two different partners in ASCENS, namely experts from swarm robotics (ULB) and experts on formal methods (ISTI). The experience obtained with the modelling and analysis of this case study may provide useful insight in what features to include into the ASCENS language SCEL.

3.4.2 Fluid-Flow Analysis

Fluid-flow analysis is another approach to deal with state-space explosion arising from models of large scale systems. The approach consists in providing a continuous-state approximation of a Markov chain by means of a system of ordinary differential equations (ODEs) in a manner which is insensitive to the population of agents that are in the system under study. For this reason, it lends itself well to modeling and analyzing the kinds of scenarios of relevance for ASCENS. In this paragraph we provide an application to robot-swarm decision making. ODE analysis will also be used in Section 5.

Fluid-Flow Analysis of Ant Colonies and Robot Swarm Decision-making
Based on the Bio-PEPA syntax, the underlying ODE model can be generated automatically and in a systematic way [Hil05a] using the Bio-PEPA tool suite [CDG+09]. This provides yet another view on the behavioural aspects of a system. One can, for example, explore numerically the sensitivity of the system to initial values and discover stationary points and other aspects related to stability analysis of the system. Such an analysis has been performed both for the emergent behaviour of an Argentine ant colony that is capable of converging on the shortest path based on a decentralised strategy involving pheromone trail laying [ML12] and, to a lesser extent, for the robot swarm decision-making strategy [MBL+12].

As an example, in Fig. 8(a) the fluid-flow analysis of the fraction of pheromones that ants lay on the long and the short path is shown. In Fig. 8(b) the relation between stochastic simulation and fluid approximation results for the robot swarm decision-making strategy is shown for the fraction of robots that have been converted to select the short path in the system over time.



(a) Fraction of pheromones on long and short path

(b) Fraction of robots voting for the short path

Figure 8: Fluid-Flow (ODE) results for an ant colony and robot swarm decision-making.

4 Compositional and Incremental Verification Techniques

Component-based design confers numerous advantages, in particular, an increased productivity through reuse of existing components. Nonetheless, establishing the correctness of the designed systems remains an open issue. In contrast to other engineering disciplines, software and system engineering badly ensures predictability at design time. Consequently, a posteriori verification as well as empirical validation are essential for ensuring correctness of designed systems.

Verifying implementations of concurrent systems is a challenging problem which is intensively studied [BR01, BR02, BCLR04, CPR06, BHJM07]. There exists solutions that consist in performing a static analysis of the code and generate an abstract mathematical model [HJMS02] on which properties are verified with powerfull tools such as SAT solvers [DdM06, MB08]. Other approaches use theorem provers [HHKR10]. Such approaches rarely scale to complete verification of large systems. We therefore propose different approach. Instead of verifying the code, we ensure that the model is correct and its correctness is preserved in the automatically generated code by our tools. We believe this is a breakthrough as it allows to take the incremental design into account, which drastically helps simplify verification. Moreover, our verification method is independent of any programming language.

Monolithic verification [CGP99, QS82] of component-based systems often requires computing for a composite component the product of its constituents by using both interleaving and synchronization. The complexity of the product system is often prohibitive due to state explosion. In a series of recent works, it has been advocated that *compositional verification techniques* can be used to cope with state explosion [Pnu85, HQR98, dRdBH$^+$00, GPB02, CGP03, CAC08, FCC$^+$08]. The key to compositional verification techniques is the application of divide-and-conquer techniques to infer global properties of complex systems from properties of their components.

Methods based on deductive techniques [MP95], or assume-guarantee techniques [Jon83, Pnu85, HQR98, dRdBH$^+$00, GPB02, CGP03, CAC08, FCC$^+$08], rely on finding an inductive invariant of a given program that is stronger than the invariant to be verified. The drawback of these techniques is that there is no good method to find such an invariant. Assume-guarantee methods always have difficulties to find decompositions into subsystems and choosing adequate assumptions for such a particular decomposition. Our method works reversely and thus avoid this problem.

A series of recent works [Lar89, AH01, dAdSF$^+$05, DLL$^+$10] propose compositional techniques based on interface theories. The conceptual idea is to check whether a component satisfies a property that is expressed by an interface modeled as an automaton. Using compositional design-based

rules, one can infer the interface that will be satisfied by the combination of the components. Interface theories permit richer logical operations than those available in our framework. However, the communication primitive that drives the composition between interfaces is not as expressive as what we propose here. Moreover, it is hard to decompose the interface representing the global properties into smaller interfaces on basic components. Finally, representing deadlock with interfaces involves a tedious task.

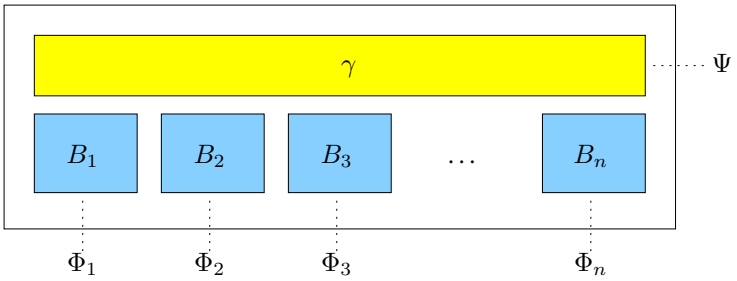


Figure 9: Compositional verification.

A compositional verification method based on invariant computation is presented in [BBNS08, BBNS09]. The method is based on the following rule (see Figure 9):

$$\frac{\{B_i\langle\Phi_i\rangle\}_i \quad \Psi \in II(\|_\gamma\{B_i\}_i, \{\Phi_i\}_i) \quad (\bigwedge_i \Phi_i) \wedge \Psi \Rightarrow \Phi}{\|_\gamma\{B_i\}_i\langle\Phi\rangle}$$

The rule allows to prove invariance of property Φ for systems obtained by using an n-ary composition operation $||$ on a set of components $\{B_i\}_i$ parameterized by a set of interactions γ. It is based on the computation of a global invariant that is the conjunction of local invariants Φ_i of constituent components B_i and an *interaction invariant* Ψ. The latter expresses constraints on the global state space induced by interactions between components. In [BBNS08], we have shown that Ψ can be computed automatically from abstractions of the system to be verified. That is, we provide an effective procedure, denoted II in the rule, which allows to compute interaction invariants from finite state abstraction of the components B_i with respect to its local invariant Φ_i.

Boolean Behavioral Constraints (BBCs) allows to relate interactions between different components with their internal transitions. BCCs are used to compute interaction invariants by applying two different symbolic techniques, namely (1) iterative compution of fixpoints, and (2) solution of a set of Boolean equations. Both techniques have been implemented and outperform the enumerative methods. It is nevertheless important to mention that none of the previously implemented methods take incremental design aspects into account. Incremental system design often works by adding new interactions to existing sets of components. Each time an interaction is added, one may be interested to verify whether the resulting system satisfies some given property. Indeed, it is important to report an error as soon as it appears. However, each verification step may be time consuming, which means that intermediate verification steps are generally avoided. This situation can be improved if the result of the verification process is reused when new interactions are added. Existing methods, including the ones in [BBNS08, BBNS10], do not focus on such aspects of modular verification.

Incremental verification method further improves the previous BBC-based method. The key idea is to reuse the already computed invariants of the constituents of a composite component in order to compute global invariants. In many situations reusing of existing invariants reduces considerably the verification effort. Incremental verification requires the formalization of the construction of hierarchically structured composite components. For instance, Figure 10 shows a composite component

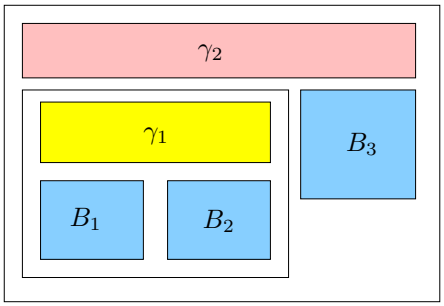


Figure 10: Incremental design.

obtained as the composition of two components by using a set of interactions γ_2. One of these components is the composition of two components by using the set of the interactions γ_1. The computation of invariants of the component $\gamma_1(B_1, B_2)$ is obtained by combining invariants of the atomic components B_1 and B_2 with an interaction invariant characterizing the restriction of the interactions γ_1 on the product of the composed components. Following the incremental construction process, invariants of $\gamma_2(B_3, \gamma_1(B_1, B_2))$ are obtained by combining invariants of $\gamma_1(B_1, B_2)$ and invariants of B_3 with an interaction invariant induced by the application of γ_2.

The method has been implemented in the D-Finder toolset [BBNS09] and applied to check deadlock-freedom on several case studies described in the BIP (Behavior, Interaction, Priority) [BBS06] language. The results of these experiments show that D-Finder is sometimes exponentially faster for checking deadlock-freedom than state-of-the-art verification tools such as NuSMV [CCGR00]. We have been able to verify deadlock-freedom and safety properties of a large part (tens of components, hundreds of interactions) of the functional level of the DALA autonomous robot [BGL+08]. It is important to notice that from the verified BIP model it is possible to generate C code. For DALA, BIP generated C code (more than 500000 lines) for application modules and their coordination at execution level. The generated C code preserves properties of the BIP model and deadlock-freedom in particular.

5 Application of the ASCENS Design Flow to the Robotics Case Study

In this section we bring together some of the techniques presented in this report and present an instance of the ASCENS design flow (see Section 1) covering requirement specification, modeling, and analysis of service-component ensembles in a unified fashion by means of formal methods. We illustrate this via of an example based on the robotics case study, summarizing a recent publication on this topic [**?**]. Here, we make use of SOTA (cf. Section 3.2) for specifying the overall domain and the requirements of an ensemble; the SCEL language is employed for the reasoning of the behavior of autonomic components; finally, based on the SCEL model, we develop an ODE model for the quantitative verification of the SOTA requirements. Model validation is done with the ARGoS simulator.

5.1 Garbage-Collecting Robots

We consider a swarm of robots that collects garbage in a rectangular exhibition hall (cf. Fig. 11). The robots should move around the room, pick up the garbage that visitors have dropped and move it to the service area. For simplicity we assume that the service area is just a rectangular strip along one side of

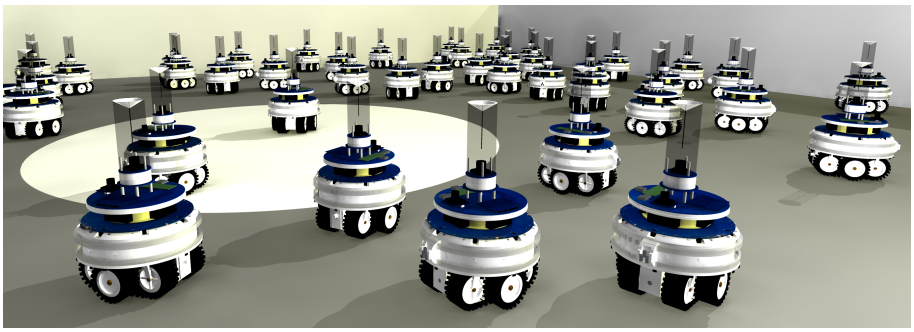


Figure 11: Ensemble of robots

the hall. Since visitors do not want to be distracted by too many robots driving around the exhibition area, there should be as few robots outside the service area as possible while still keeping the hall adequately clean. Furthermore, for environmental and cost reasons the robots should minimize the amount of energy that the swarm consumes. Depending on the sensors of the robots and the type of garbage they are collecting, they may be able to perceive garbage from some distance away, or they may be perceive garbage only while they are driving over it.

5.2 Requirements Specification with SOTA

Here we consider a simple SOTA model of the robot ensemble, under the assumption that we have a fixed number N of robots, might be defined as follows: The state space consists of the states of the individual robots, a count $g^\sharp$ of the items of garbage currently in the public part of the exhibition area, and a boolean flag $o^\flat$ that indicates whether the exhibition is currently open for the public or not. We describe each robot by its position in the exhibition area, p_i, and its state s_i. The state can either be Resting, Searching, or Carrying, depending on whether the robot is currently resting, searching for garbage, or carrying a garbage item back to the service area.

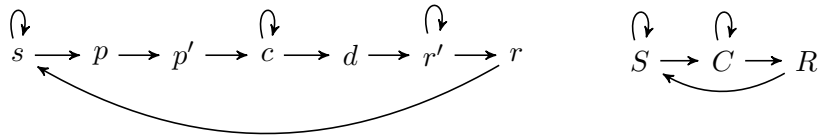
Accordingly, the state of the affairs space of the robot ensemble can be described as follows:

p_i	$= \langle x_i, y_i \rangle \in \mathbb{R} \times \mathbb{R}$	Position of robot i
Area	$\subseteq \mathbb{R} \times \mathbb{R}$	Exhibition Area
s_i	$\in \{\mathsf{Searching}, \mathsf{Resting}, \mathsf{Carrying}\}$	State of robot i
$g^\sharp$	$\in \mathbb{N}$	Number of garbage items
$o^\flat$	$\in \mathbb{B}$	Exhibition open for public?
$\mathbf{Q}$	$= \{\langle p_1, s_1, \ldots, p_N, s_N, g^\sharp, o^\flat \rangle \mid p_i \in \mathsf{Area}\}$	State space

One of our goals might be to always have fewer than 300 garbage items in the exhibition area while the exhibition is open. This could be described by the following goal G^1:

$$G^1_{pre} \equiv o^\flat = true, \qquad G^1_{maintain} \equiv g^\sharp < 300, \qquad G^1_{post} \equiv o^\flat = false.$$

G^1 states that whenever the exhibition opens (i.e., $o^\flat$ becomes $true$), the number of garbage items on the floor, $g^\sharp$, is less than 300. Once the exhibition closes, the postcondition of the goal becomes $true$ and the goal is abandoned until the exhibition opens again.



(a) Transition system from the SCEL model. (b) Reduced model.

Figure 12: Qualitative discrete-state behavior of a garbage-collecting robot.

5.3 SCEL model of the case study

Qualitatively, the behavior of a single robot could be modeled with the following SCEL fragment.

$$
\begin{aligned}
s &\triangleq \mathbf{get}(collision)@ctl.s + \mathbf{get}(item)@ctl.p \\
p &\triangleq \mathbf{get}(items, !x)@master.p' \\
p' &\triangleq \mathbf{put}(items, x+1)@master.c \\
c &\triangleq \mathbf{get}(collision)@ctl.c + \mathbf{get}(arrived)@ctl.d \\
d &\triangleq \mathbf{put}(dropped)@master.r' \\
r' &\triangleq \mathbf{get}(collision)@ctl.r' + \mathbf{put}(sleep)@timer.r \\
r &\triangleq \mathbf{get}(elapsed)@timer.s
\end{aligned}
$$

The process constants stand for: s = searching for a garbage item; p = picking up item; c = carrying the item (returning to service area); d = dropping off item at the service area; r' = searching for a rest place; r = resting, to reduce power consumption and avoid robot overcrowding at the exhibition hall. Processes s, c, and r' exhibit similar behavior in that they may consume a tuple $collision$ which is produced by some controller ctl, which leads to a change in direction of the robot movement, while not changing the high-level behavior of the process. The controller may also produce an $item$, in which case process s behaves then as p. Here, we assume a central repository $master$ which keeps track of the total number of items collected during the evolution of the system. The current value is first retrieved and then put back into the repository after being incremented. Process r' puts an item $sleep$ into the tuple space of a $timer$. It will be able to resume when the timer puts an $elapsed$ tuple in its tuple space. The labeled transition system for this process is shown in Fig. 12(a). For reasons of space, the (obvious) transition labels are not explicitly given.

The names ctl, $master$, and $timer$ are assumed to be exposed by other components, k, m, t, respectively (not shown here for brevity).

For the purposes of quantitative evaluation, the behavior may be simplified by making the following assumptions: (i) The transitions $p \to p'$ and $p' \to c$ take up a negligible amount of time with respect to the representative time scales of the system; similarly, (ii) the durations of $d \to r'$ and $r' \to r$ are assumed to be negligible. In other words, (i) and (ii) imply that the robot goes to sleep as soon as it enters the service area.

The validity of such assumptions was successfully validated with the simulation experiments with ARGoS (cf. Section 3.1) . Overall, these simplifications lead to a smaller discrete-state description as shown in Fig. 12(b). Notice that the three states of this reduced labelled transition system correspond to the states of the robot described in SOTA in Section 5.2.

	S	C	R
Simulation	15.972	3.778	0.250
Model	16.070	3.730	0.200

Table 1: Model validation. Steady-state ODE estimates of robot sub-populations against discrete-event simulation of the system with ARGoS.

5.4 Quantitative Analysis and Validation

As discussed in Section 3.4.2, a Markovian model may suffer when the number of robots is high. Thus, we consider the following system of ODEs, in a manner which is similar to Bio-PEPA:

$$\dot{S} = -\mu SG/(S+C+G) + \beta R, \qquad \dot{C} = +\mu SG/(S+C+G) - \gamma C,$$
$$\dot{R} = +\gamma C - \beta R, \qquad \dot{G} = +\lambda - \mu SG/(S+C+G).$$

We briefly comment on the term $\mu SG/(S+C+G)$ which is crucial for understanding the dynamics of the system; a full model explanation is available in [**?**]. The parameter μ is intended to be the *encounter rate* of each robot, i.e., the opposite of the average time between collisions between robots or between a robot and an item. The fraction $G/(S+C+G)$ represents the probability of a successful encounter, which is given as the ratio of items with respect to the total amount of objects a robot may encounter. The factor S is the multiplicative factor in order to account for the whole population of exploring robots. The other parameters may be interpreted as follows: λ is the rate at which garbage is dropped onto the exhibition hall; γ is the speed at which a robot returns to the service area; $1/\beta$ is the rest time at the service area.

To check the validity of the model, a discrete-event simulation of the system under study was developed with ARGoS. The source code for a robot controller was instrumented to record the timestamps of transitions according to the classification of states in Fig. 12(b). With an arena size of 16 squared meters with $N = 20$ robots, these parameters were found to be $\mu = 0.012$ and $\gamma = 0.003$. Across all experiments, we set $\lambda = 0.010$ and $\beta = 0.050$. The simulation also logged the total number of robots in each of the states S, C, R as a function of time. The mean steady-state estimates were calculated by using 150 independent runs of the simulation, each lasting ten hours of simulated time.

The results of this validation, shown in Table 1, demonstrate a very good accuracy of the model, with a maximum error less than 0.5% relative to the total population of robots.

5.5 Quantitative Verification of SOTA Requirements

The SOTA requirement in Section 5.2 of having less than 300 garbage items in the arena can be translated into a sensitivity analysis which looks at the estimated value of $G(t)$, solution of the system of ODEs. For example, this can be done by inspecting a curve which plots the steady-state *throughput*, (i.e., the rate at which garbage items are picked up, which can be formally derived from the ODE solution), against the average rest time $1/\beta$, as shown in Fig. 13. The throughput curve, in solid line, shows insensitivity for a wide range of rest times, until about 1500. This is because, in those situations, the arena is kept relatively clean (with about 2 garbage items, dash-dotted line), therefore many robots keep exploring but they encounter an item infrequently. As rest times are further increased, however, the robots cannot keep up with the waste; throughput decreases because fewer robots are present, and the arena becomes more soiled. Thus, the maximum allowed rest time predicted by the model, corresponding to the lowest energy consumption possible whilst achieving the desired *maintain goal*, corresponds to 1580, when the garbage-item line and the maintain-goal one (dotted line) intersect.

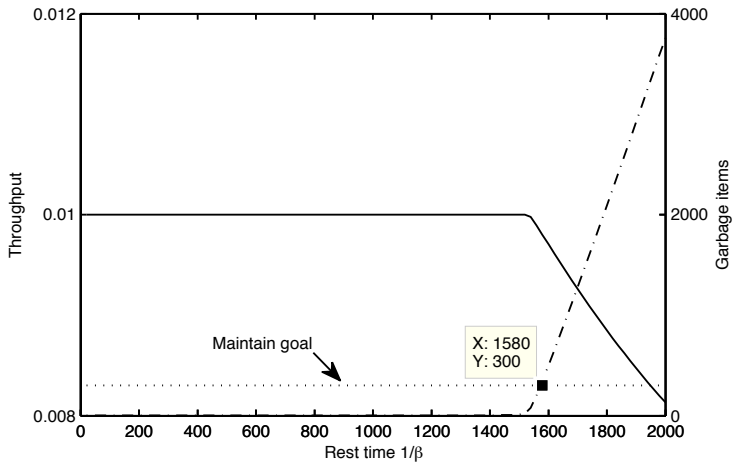


Figure 13: Sensitivity analysis of throughput of garbage collection (solid line) and dirtiness of exhibition hall (dash-dotted line) against rest time at the service area. The dotted line shows the SOTA requirement (*maintain goal*) that there must not be more than 300 garbage items in the arena.

6 Conclusion

We have presented the ASCENS design flow that aims for specifying, validating and implementing Service Component Ensembles (SCEs). Validation is one of the main challenges of this design flow since it has to accommodate with the complexity of SCEs due to (self-)adaptive behavior, uncertain/changing environment, dynamicity, very large number of components, etc. It requires adequate representations corresponding to system models derived from abstract specifications. It also needs the development of specific analysis methods targeting both efficiency and accuracy with respect to the properties of interest. The presented methods and corresponding models fall in three categories: simulation, quantitative analysis, and (compositional) verification. One of the techniques used to achieve the analysis of SCEs is abstraction. It allows building simplier models from more complex ones. This mechanism is used for instance by the ARGoS simulator in which the physics is abstracted to speed-up simulations allowing real-time simulation of swarm robotic applications. In statistical/probabilistic model checking abstractions are obtained by means of non-deterministic behavior with associated probability. We explained how the proposed methods can apply to SCEs, and illustrated them with the ASCENS case studies.

We have also outlined in Section 5 a full development cycle for large-scale service-component ensembles which covers requirements, specification, and analysis assisted by the use of formal methods, illustrated on a robotics case study. This is only a possible instance of the ASCENS design flow, and other techniques developed with the context of this project may be considered in lieu of or in addition to those discussed in this section. For instance, the ODE analysis could be complemented with stochastic simulation (as done in Section 3.4) and the evaluation by means of sensitivity analysis could be replaced by (quantitatively) model checking a logical formula that encodes the SOTA requirements. Although the links between SOTA, SCEL, and ODEs have been established in an ad-hoc manner for this specific case study, there is ample room for mechanizing this process; especially, with regards to the automatic generation of quantitative models from (a suitable fragment of) a stochastic extension of SCEL.

Regarding security we plan to work next year in the following direction. We will use information flow policies which specify end-to-end confidentiality and integrity requirements by puting global

constraints on the information flow. We will mainly focus on non-interference properties, that is, confidential data must not affect the publicly visible behavior of the system. We propose to associate information flow at service component level for automating the construction of secure systems with security policy verification and code generation. The developer will focus on the functional part of his code and specify the security policy of his system at architecture configuration level. Component information flow compiler will check that no inerference problem exists and generate the security code corresponding to the specified configuration. The verification of the security will be done at intra-component level based on the behavior of the components, and at inter-component level based on the inter-connections (i.e. possible interactions) between the components.

A Non-Ascens Tools used for Verification

A.1 PEPA

PEPA is a process algebra for the performance evaluation of computer systems [Hil96]. It is based on the following two-level grammar:

$$S ::= (\alpha, r).S \mid S + S \mid A, \qquad \text{with } A = S$$
$$P ::= S \mid P\backslash L \mid P \underset{L}{\bowtie} P$$

The language enables the specification of *sequential components* (first line), which are capable of doing an activity via the *prefix* operator $(\alpha, r).S$. The activity consists of an action label, α, and a value, r, which is interpreted as the process' capacity, the maximum rate at which it can perform an action if no synchronisation occurs [Hil94]. The choice operator $S + S$ enables multiple activities. Recursion, typically used to allow cyclic behaviour, is obtained by means of constants, via assignments $A{=}S$. *Composite processes* (second line) are formed by sequential components. Hiding, $P\backslash L$ affects the action labels which are visible to the environment. Cooperation $P \underset{L}{\bowtie} P$ is used to synchronise two processes over the set of shared action labels. For example, $(\alpha, r_1).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, r_2).P_2$ can be shown to yield a transition to process $P_1 \underset{\{\alpha\}}{\bowtie} P_2$, via an activity (α, r), with $r = \min\{r_1, r_2\}$. The semantics of PEPA, which assigns the minimum of the two capacities to synchronised transitions, can be shown to accurately describe communication between processes over full-duplex media, e.g., Ethernet-like connections [TT12].

PEPA was originally given an operational semantics which interprets the resulting labelled transition system as a continuous-time Markov chain. In addition, it is the first process algebra to be equipped with a fluid semantics [Hil05b], which compactly represents the time-course evolution of large populations of identical processes as a system of first-order coupled differential equations. The fluid semantics, and extensions thereof (e.g., [HB10, TGH12]) are implemented in freely available software tools [TDG09, SHB11].

A.2 Bio-PEPA

Bio-PEPA [CH09] is a process algebraic language that was originally developed for the stochastic modelling and analysis of biochemical systems. Bio-PEPA specifications consist of two main kinds of components. The first kind is called the *"species" component*, specifying the behaviour of individual entities. The second kind is the *model component*, specifying the interactions between the various species. In the context of this paper, the individual entities are the robots, and the model component defines how they interact.

The basic syntax of Bio-PEPA components is defined as:

$$S ::= (\alpha, \kappa)\ \mathtt{op}\ S \mid S + S \mid C \ \text{ with } \mathtt{op} = \downarrow \mid \uparrow \mid \oplus \mid \ominus \mid \odot \ \text{ and } P ::= P \underset{\mathcal{L}}{\bowtie} P \mid S(x)$$

where S is a *species component* and P is a *model component*.

The *prefix combinator* "op" in the prefix term $(\alpha, \kappa)\ \mathtt{op}\ S$ represents the impact that action α has on species S. Specifically, $\downarrow$ indicates that the number of entities of species S reduces when α occurs, and $\uparrow$ indicates that this number increases. The amount of the change is defined by the stoichiometry coefficient κ. This coefficient captures the multiples of an entity involved in an occurring action. The default value of κ is 1, in which case we simply write α instead of (α, κ). Action durations are assumed to be random variables with negative exponential distributions, characterised by their *rates*. The rate of action α is defined by a so called functional rate or kinetics rate. Action rates are defined in the context section of a Bio-PEPA specification.

The symbol $\oplus$ denotes an *activator*, $\ominus$ an *inhibitor* and $\odot$ a generic *modifier*, all of which play a role in an action without being produced or consumed and have a defined meaning in the biochemical context.

The operator "+" expresses the choice between possible actions, and the constant C is defined by the equation $C{=}S$. The process $P \underset{\mathcal{L}}{\bowtie} Q$ denotes synchronisation between components P and Q, the set $\mathcal{L}$ determines those actions on which the components P and Q are forced to synchronise. The shorthand $P \bowtie Q$ denotes synchronisation on all actions that P and Q have in common. In $S(x)$, the parameter $x \in \overset{*}{\mathbb{R}}$ represents the initial amount of the species.

Bio-PEPA has been extended with a notion of *discrete locations* in which populations of components can reside and move between. A Bio-PEPA *system* with *locations* consists of a set of species components, a model component, and a context containing definitions of locations, functional/kinetics rates, parameters, etc.. The prefix term $(\alpha, \kappa)\ \mathtt{op}\ S@l$ is used to specify that the action is performed by S in location l.

Bio-PEPA is given a formal operational semantics [CH09] which is based on Continuous Time Markov Chains (CTMCs). It is supported by a suite of software tools which automatically process Bio-PEPA models and generate internal representations suitable for different types of analysis [CH09, CDG+09]. These tools include mappings from Bio-PEPA to differential equations (ODE) supporting a fluid flow approximation [Hil05a], stochastic simulation models [Gil77], CTMCs with levels [CH08] and PRISM models [KNP11] amenable to statistical model checking. Consistency of the analyses is supported by a rich theory including process algebra, and the relationships between CTMCs and ODE.

A.3 CIAO Language

This section briefly presents the CIAO language. Complete information about it can be found at the web page `http://www.ciaohome.org/`, where it is possible to find the latest version and its documentation. In the following we just sum up the main features of the language.

CIAO is a multiparadigm programming language offering an advanced programming environment. It offers a high-performance, freely available Prolog system, supporting ISO-Prolog, and at the same time, its modular design allows both restricting and extending the basic language through libraries. Also via libraries, CIAO supports functional, higher-order, as well as object-oriented programming styles. Therefore, modules in ISO-Prolog can be freely combined with other modules written in these other supported paradigms.

CIAO offers a rich programming environment. It, besides a classical top-level, includes a rich development interface based on Emacs as well as an embeddable source-level debugger and several execution visualization tools. Moreover, the programming environment provides an automatic documentation generator for LP/CLP programs.

References

[AH01] Luca De Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *EMSOFT*, pages 148–165. Springer-Verlag, 2001.

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking Continuous Time Markov Chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[BBB+11] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the bip framework. *IEEE Software*, 28(3):41–48, 2011.

[BBB+12] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. *STTT*, 14(1):53–72, 2012.

[BBBS08] Ananda Basu, Philippe Bidinger, Marius Bozga, and Joseph Sifakis. Distributed semantics and implementation for systems with interaction and priority. In Suzuki et al. [SHYEF08], pages 116–133.

[BBL+10] Saddek Bensalem, Marius Bozga, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. Incremental component-based construction and verification using invariants. In Bloem and Sharygina [BS10], pages 257–256.

[BBNS08] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. Compositional verification for component-based systems and application. In *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.

[BBNS09] S. Bensalem, M. Bozga, T-H. Nguyen, and J. Sifakis. D-Finder: A tool for compositional deadlock detection and verification. In *Proceedings of the 21st International Conference on Computer Aided Verification*, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.

[BBNS10] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. Compositional verification for component-based systems and application. *IET Software*, 4:179–235, 2010.

[BBQS12] Saddek Bensalem, Marius Bozga, Jean Quilbeuf, and Joseph Sifakis. Knowledge-based distributed conflict resolution for multiparty interactions and priorities. In Giese and Rosu [GR12], pages 118–134.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[BCC+97] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The ciao prolog system. Reference manual. Technical Report CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), 1997.

[BCLR04] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft. In *In: IFM. (2004*, pages 1–20. Springer, 2004.

[BGL+08] Ananda Basu, Matthieu Gallien, Charles Lesire, Thanh-Hung Nguyen, Saddek Bensalem, Félix Ingrand, and Joseph Sifakis. Incremental component-based construction and verification of a robotic system. In Ghallab et al. [GSFA08], pages 631–635.

[BHJM07] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker BLAST: Applications to software engineering. *STTT*, 9(5-6):505–525, 2007.

[BJMS12] Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. Modeling dynamic architectures using dy-bip. In Gschwind et al. [GPGB12], pages 1–16.

[BJS10] Marius Bozga, Mohamad Jaber, and Joseph Sifakis. Source-to-source architecture transformation for performance optimization in bip. *IEEE Trans. Industrial Informatics*, 6(4):708–718, 2010.

[BKH99] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In *Concur '99* [CON99], pages 146–162.

[BMR95] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *IJCAI (1)*, pages 624–630, 1995.

[BMR01] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based contstraint logic programming: syntax and semantics. *ACM TOPLAS*, 23(1):1–29, 2001.

[BMRS10] S. Bistarelli, U. Montanari, F. Rossi, and F. Santini. Unicast and multicast QoS routing with soft-constraint logic programming. *ACM TOCL*, 12(1):5, 2010.

[BR01] Thomas Ball and Sriram K. Rajamani. The SLAM toolkit. In *CAV*, pages 260–264, 2001.

[BR02] Thomas Ball and Sriram K. Rajamani. The SLAM project: debugging system software via static analysis. In *POPL*, pages 1–3, 2002.

[BS10] Roderick Bloem and Natasha Sharygina, editors. *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*. IEEE, 2010.

[CAC08] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Transactions on Software Engineering and Methodology*, 17(2):1–52, 2008.

[CCGR00] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2:410–425, 2000.

[CDG+09] F. Ciocchetta, A. Duguid, S. Gilmore, M. L. Guerriero, and Hillston J. The Bio-PEPA Tool Suite. In *Proc. of the 6th Int. Conf. on Quantitative Evaluation of SysTems (QEST 2009)*, pages 309–310, Washington, DC, USA, 2009. IEEE Computer Society.

[CES09] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, 2009.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. The MIT Press, 1999.

[CGP03] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.

[CH08] F. Ciocchetta and J. Hillston. Bio-PEPA: An extension of the process algebra PEPA for biochemical networks. *ENTCS*, 194(3):103–117, 2008.

[CH09] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *TCS*, 410(33-34):3065–3084, 2009.

[CON99] *Concur '99*, volume 1664 of *LNCS*. Springer, Berlin, Heidelberg, 1999.

[CPR06] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Terminator: Beyond safety. In *CAV*, pages 415–418, 2006.

[dAdSF+05] Luca de Alfaro, Leandro Dias da Silva, Marco Faella, Axel Legay, Pritam Roy, and Maria Sorea. Sociable interfaces. In *FroCos*, pages 81–105, 2005.

[DBL11] *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*. IEEE Computer Society, 2011.

[DdM06] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the 18th Computer-Aided Verification conference*, volume 4144 of *LNCS*, pages 81–94. Springer-Verlag, 2006.

[DLL+10] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *HSCC*, pages 91–100, 2010.

[dRdBH+00] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods.* Cambridge University Press., New York, NY, USA, 2000.

[EJL+03] Johan Eker, Jrn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach, 2003.

[FCC+08] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending automated compositional verification to the full class of omega-regular languages. In *TACAS*, pages 2–17, Berlin, Heidelberg, 2008. Springer-Verlag.

[GADP89] S. Goss, S. Aron, Deneubourg, J.-L., and J. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76:579–581, 1989.

[Gil77] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[GMW97] David Garlan, Robert Monroe, and David Wile. Acme: An architecture description interchange language. In *in Proceedings of CASCON97*, pages 169–183, 1997.

[GPB02] Dimitra Giannakopoulou, Corina S. P"s"reanu, and Howard Barringer. Assumption generation for software component verification. In *ASE '02: Proceedings of the 17th IEEE international conference on Automated software engineering*, pages 3–12, Washington, DC, USA, 2002. IEEE Computer Society.

[GPGB12] Thomas Gschwind, Flavio De Paoli, Volker Gruhn, and Matthias Book, editors. *Software Composition - 11th International Conference, SC 2012, Prague, Czech Republic, May 31 - June 1, 2012. Proceedings*, volume 7306 of *Lecture Notes in Computer Science*. Springer, 2012.

[GR12] Holger Giese and Grigore Rosu, editors. *Formal Techniques for Distributed Systems - Joint 14th IFIP WG 6.1 International Conference, FMOODS 2012 and 32nd IFIP WG 6.1 International Conference, FORTE 2012, Stockholm, Sweden, June 13-16, 2012. Proceedings*, volume 7273 of *Lecture Notes in Computer Science*. Springer, 2012.

[GSFA08] Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors. *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.

[HB10] Richard A. Hayden and Jeremy T. Bradley. A fluid analysis framework for a Markovian process algebra. *Theor. Comput. Sci.*, 411(22-24):2260–2297, 2010.

[HHKR10] Thomas A. Henzinger, Thibaud Hottelier, Laura Kovs, and Andrey Rybalchenko. Aligators for arrays. In *LPAR*, TACAS 2001, pages 348–356, 2010.

[Hil94] Jane Hillston. The nature of synchronisation. In *Proceedings of the Second International Workshop on Process Algebras and Performance Modelling*, pages 51–70, 1994.

[Hil96] J. Hillston. *A compositional approach to performance modelling*. Cambridge University Press, New York, NY, USA, 1996.

[Hil05a] J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the 2th International Conference on Quantitative Evaluation of SysTems (QEST 2005)*, pages 33–43, Washington, DC, USA, 2005. IEEE Computer Society.

[Hil05b] J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, pages 33–43, Torino, Italy, September 2005. IEEE Computer Society Press.

[HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '02, pages 58–70, New York, NY, USA, 2002. ACM.

[Hol91] Gerard J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[HQR98] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 440–451, London, UK, 1998. Springer-Verlag.

[HZWS12] N. Hoch, K. Zemmer, B. Werther, and R. Y. Siegwarty. Electric vehicle travel optimization - customer satisfaction despite resource constraints. In *IEEE IVS*, 2012.

[JL87] J. Jaffar and J. L. Lassez. Constraint logic programming. In *POPL*, pages 111–119. ACM Press, 1987.

[Jon83] Cliff B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd Int. Conf. on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, Heidelberg, 2011.

[Lar89] Kim Guldstrand Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, pages 232–246, 1989.

[MB08] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[MBL+12] M. Massink, M. Brambilla, D. Latella, M. Dorigo, and M. Birattari. Analysing robot swarm decision-making with Bio-PEPA. In *Proc. of the International Swarm Intelligence Conference ANTS 2012*, volume To appear of *LNCS*. Springer, 2012.

[MFS+11] M. A. Montes de Oca, E. Ferrante, A. Scheidler, C. Pinciroli, M. Birattari, and M. Dorigo. Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. *Swarm Intelligence*, 5(3–4):305–327, 2011.

[MK96] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In *In Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1996.

[ML12] M.Massink and D. Latella. Fluid analysis of foraging ants. In M. Sirjani, editor, *Coordination Models and Languages*, volume 7274 of *LNCS*, pages 152–165. Springer-Verlag, 2012.

[MM12] G.. Monreale and U. Montanari. Soft constraint logic programming for electric vehicle travel optimization. In *To appear in WLP*, 2012.

[MP92] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[MP95] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. *Logics and models of concurrent systems*, F13:123–144, 1985.

[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. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 5027–5034. IEEE Computer Society Press, Los Alamitos, CA, September 2011.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351. Springer-Verlag, 1982.

[SHB11] Anton Stefanek, Richard A. Hayden, and Jeremy T. Bradley. Gpa - a tool for fluid scalability analysis of massively parallel systems. In *QEST* [DBL11], pages 147–148.

[SHYEF08] Kenji Suzuki, Teruo Higashino, Keiichi Yasumoto, and Khaled El-Fakih, editors. *Formal Techniques for Networked and Distributed Systems - FORTE 2008, 28th IFIP WG 6.1 International Conference, Tokyo, Japan, June 10-13, 2008, Proceedings*, volume 5048 of *Lecture Notes in Computer Science*. Springer, 2008.

[TDG09] M. Tribastone, A. Duguid, and S. Gilmore. The PEPA Eclipse Plug-in. *Performance Evaluation Review*, 36(4):28–33, March 2009.

[TGH12] Mirco Tribastone, Stephen Gilmore, and Jane Hillston. Scalable differential analysis of process algebra models. *IEEE Trans. Software Eng.*, 38(1):205–219, 2012.

[Tsa93] E. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.

[TT12] Max Tschaikowski and Mirco Tribastone. Generalised Communication for Interacting Agents. In *9th International Conference on Quantitative Evaluation of Systems*, September 2012. To appear.

[YKNP06] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.