QUANTICOL

A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours

quanticol
http://www.quanticol.eu

D4.1

CAS-SCEL language design

Revision: 1.0; March 31, 2014

Author(s): Luca Bortolussi (CNR), Rocco De Nicola (IMT), Cheng Feng (UEDIN), Vashti Galpin (UEDIN), Jane Hillston (UEDIN), Diego Latella (CNR), Michele Loreti (IMT), Mieke Massink (CNR), Valerio Senni (IMT),

Due date of deliverable: Month 12 (March 2014)
Actual submission date: March 31, 2014
Nature: R. Dissemination level: PU

Funding Scheme: Small or medium scale focused research project (STREP)
Topic: ICT-2011 9.10: FET-Proactive 'Fundamentals of Collective Adaptive Systems' (FOCAS)
Project number: 600708

Coordinator: Jane Hillston (UEDIN)
e-mail: `Jane.Hillston@ed.ac.uk`
Fax: +44 131 651 1426

Part. no.	Participant organisation name	Acronym	Country
1 (Coord.)	University of Edinburgh	UEDIN	UK
2	Consiglio Nazionale delle Ricerche – Istituto di Scienza e Tecnologie della Informazione "A. Faedo"	CNR	Italy
3	Ludwig-Maximilians-Universität München	LMU	Germany
4	Ecole Polytechnique Fédérale de Lausanne	EPFL	Switzerland
5	IMT Lucca	IMT	Italy
6	University of Southampton	SOTON	UK
7	Institut National de Recherche en Informatique et en Automatique	INRIA	France

COOPERATION

Executive summary

We report on the progress made with the development of CAS-SCEL for what concerns its *design principles* and the identification of *primitives* (such as movement primitives or space abstraction primitives) and *interaction patterns* (such as broadcast communication or anonymous interaction) that are needed in the case studies and, more generally, in Collective Adaptive Systems (CAS) design. Our first concern has been the identification of abstractions and linguistic primitives for collective adaptation, location modelling, knowledge handling, and system interaction and aggregation.

To this purpose we have taken as starting point a number of exploratory formalisms that are based on, or have taken inspiration from PEPA and SCEL, two languages that partners of the project have developed in the past years and that have proved very successful in modelling adaptive systems (SCEL) and in supporting quantitative analysis (PEPA). We use four exploratory formalisms, StocS, PALOMA, PEPA-S and Stochastic-HYPE, with specific features that are each very interesting for CAS modelling and analysis and assess the impact of new primitives on CAS specification and verification, by considering a concrete scenario, inspired by the bike-sharing case study.

Each of the exploratory languages has specific traits. One of the key features of StocS is the use of attribute-based communication that is a valuable alternative to broadcast or binary synchronisation that appear to be inappropriate in CAS and fits well with the notions of *anonymity* and *dynamicity* of CASs. PALOMA, instead, stresses the role of locations as attributes of agents; their communication abilities depend on their location, through a *perception function* and only agents who enable the appropriate reception action have the capability to receive the message. In PEPA-S heterogeneous populations of indistinguishable agents operating on a set of locations are considered. PEPA-S aims at distilling and studying the set interaction patterns that are typical of CASs. Like in PALOMA, in PEPA-S the ability of agents to communicate depends on their location. Stochastic HYPE aims at modelling three distinct types of behaviour: instantaneous events that happen as soon specific conditions are met, stochastic events with durations drawn from exponential distributions and continuous behaviour described by ODEs over systems variables.

In this report, we first present the general and desired features of modelling languages for CAS, then we use the four different formalisms to model and analyse the running example based on city bike sharing case study. Each language has a dedicated section that ends with an assessment with respect to the desired features introduced in the first part of the report. A concluding section summarises our contribution and describes the road map for the second year of the project.

Contents

1 Introduction

One of the main aims of WP4 is the design of a programming/specification language, that we named CAS-SCEL, to be used to model, program and analyse collective adaptive systems. It has been considered important that the language offers the possibility of integrating behavioural description and knowledge management and provides specific abstractions or linguistic primitives for key concepts like knowledge, behaviour, aggregation, and interactions. Moreover, it is essential that the language provides different kinds of interaction patterns to take into account the different communication and synchronisation capabilities of CAS and to support the multilayer structure of collective systems. Another important aspect is that the language is mathematically founded to enable both qualitative and quantitative analysis in a scalable manner.

The main purpose of the CAS-SCEL is to support modelling and programming of CAS systems at different levels of abstraction, with the objective of defining a comprehensive framework for the development, verification, and analysis of CAS. Therefore, the language design is focused on identifying the appropriate compromise between expressiveness and minimality, as well as at establishing formal foundations to guarantee feasible verification of both quantitative and qualitative properties.

In this report we shall present our first investigation in this direction and shall be concerned with the identification of abstractions and linguistic primitives for collective adaptation, location modelling, knowledge handling, and system interaction and aggregation. To this purpose we have taken as starting point a number of exploratory formalisms that are based on, or have taken inspiration by PEPA [15] and SCEL [8], two languages that partners of the project have developed in the past years and that have proved very successful in modelling adaptive systems (SCEL) and in supporting quantitative analysis (PEPA).

We have planned to assess the impact of new primitives on CAS specification and verification, and to address scalability issues, by using the exploratory formalisms to model and analyse some of the case studies (city bike-sharing, smart transportation systems, smart energy grid) of the project. In particular, in this document we consider a concrete scenario, inspired by the bike-sharing case study, and use all the exploratory formalisms to model it and to discuss the kind of analysis that can be performed. This experiment will lay the bases for a first assessment of the different options and will guide us in the work for the following years.

The four exploratory specification languages we use are StocS, PALOMA, PEPA-S and Stochastic-HYPE that have specific, distinguishing features that we consider relevant for CAS specification and verification.

- SCEL (Software Component Ensemble Language) [8], is a kernel language that has been designed to support the programming autonomic computing systems. The stochastic variant of SCEL, that we call StocS, is a first step towards the investigation of the impact of different stochastic semantics for autonomic processes, that relies on stochastic output semantics, probabilistic input semantics and on a probabilistic notion of knowledge. Moreover, StocS can be used to experiment with attribute based communication (one of the distinguishing features of SCEL) in the framework of CAS where neither broadcast nor binary synchronisation appear to be appropriate. On the contrary StocS communication primitives fit well with the notions of *anonymity* and *dynamicity* of CASs. StocS is presented in Section 3.

- In the process algebra PALOMA we have been investigating a model based on located Markovian agents [4]. Each agent in the process algebra is parameterised by a location, which can be regarded as an attribute of the agent. The ability of agents to communicate depends on their location, through a *perception function*. This can be regarded as an example of a more general class of attribute-based communication mechanisms, but for the QUANTICOL project and the transport case studies particularly, the location attribute is perhaps the most relevant. The communication is based on a multicast, as only agents who enable the appropriate reception action have the ability to receive the message. Moreover the effectiveness of the communication is subsequently adjusted according to the perception function. In all respects the language is kept simple. Movement is not distinguished, but can be modelled (even synchronised movement) as a state change which may induce change in other agents. More detail is given in Section 4.

- PEPA-S is an *ongoing* attempt to extend the stochastic process algebra PEPA [15] in a spatial context, adopting the spatial interaction mechanism developed in [4] for Markovian agent models. In PEPA-S heterogeneous populations of indistinguishable agents operating on a set of locations are considered. PEPA-S aims at distilling and studying the set interaction patterns that are typical of CASs. Indeed, even if this proposed specification language is based on PEPA, the proposed ideas can be applied to other process specification languages. Like in PALOMA, in PEPA-S the ability of agents to communicate depends on their location. Additional details about PEPA-S are provided in Section **??**.
- Stochastic HYPE is a process algebra developed to model three distinct types of behaviour: instantaneous events that happen as soon as their conditions are met, stochastic events that have durations drawn from exponential distributions (although other distributions can be modelled by using timers and instantaneous events), and continuous behaviour described by ODEs over the variables of the system. It provides a very expressive formalism that allows for modelling of space in a discrete or continuous fashion. More detail is given in Section 6.

The rest of the report is organised as follows. In the next section we shall present the general and desired features of modelling languages for CAS and shall introduce the running example based on the city bike sharing case study that will be used throughout the report to assess our linguistic proposals. In the subsequent sections we will present the key features of the different exploratory formalisms; each language presentation will be followed by the modelling of the running example and by a discussion aiming to assess its usability in the CAS framework both from the descriptive and from the analytic point of view. A concluding section summarises our contribution and describes the road map for the second year of the project.

2 Patterns and Primitives for CAS

In the initial phase of the project we have started analysing three major case studies (see Deliverable 5.1) that have been the main source of inspiration for both the design of new constructs and for their assessment. In this section we first sketch the three different scenarios we have been considering, then we outline the design principles we have derived from them.

2.1 Three Scenarios

The case studies we have considered are mainly concerned with the issue of designing smart cities that will improve improve the quality of life of citizens. The focus is on problems connected with energy consumption and transportation and with citizens mobility. In particular we have:

Smart transportation systems: These aim at enriching public transportation systems with an information technology infrastructure in order to make the service more usable and friendly for passengers, as well as more adaptive and robust with respect to traffic congestion and unpredicted changes.

Smart energy grids: These have the objective of providing optimal control over energy production, consumption, and distribution. Instantaneous supply of electricity must always meet the constantly changing demand; thus forecasting production and demand is of central importance just like the ability to influence user consumption by means of incentives or by guaranteeing partial control over appliances.

City bike sharing: These schemes can help in reducing vehicular traffic, pollution, and energy consumption. The main challenges to face when managing these systems are the reduction of costs of bike redistribution among parking stations and the maximisation of user satisfaction by appropriately locating parking stations and by guaranteeing availability of bikes and parking slots.

All three scenarios provide evidence that it is necessary to model spatial aspects, patterns of user behaviours, to handle events that can cause abnormal conditions, and need to be handled via re-routing or user incentives. It is clear that the modelling language should support the design of *adaptive behaviours* within patterns of variability, the use of *spatially dependent incentives* to shape the service request, and the gathering of *information about service usage* and users *preferences*. In terms of analysis, the modelling language should also allow *capacity planning* and *performance analysis*, as well as to study the behaviour of the system in presence of unexpected events and to provide control strategies to ensure efficient emergent behaviour.

2.2 Design Principles for CAS-SCEL

This subsection reports on the progress made on CAS-SCEL with respect to *design principles* as well as on the identification of *primitives* (such as movement primitives or space abstraction primitives) that are needed in the case studies and, more generally, in Collective Adaptive Systems (CAS) design.

Modelling languages for CAS should support *modularity* and *separation of concerns*, and should be *expressive* enough to allow succinct description of complex models. We have identified some general desirable features of modelling languages for CAS, which we briefly summarise below.

Automated Derivation of the Model. The model of interest (e.g. a Markov process, or a set of ordinary or partial differential equations) should be derivable automatically from a formally specified high-level language. Moreover, different kinds of models (such as approximate ones) should be derived in the same automated manner.

Agents + Environment view. The key components of CAS models are the *individuals* (also called *agents*) that compose the collectivity and the *environment* they exist in. Modelling the environment as an individual could be restrictive. The environment, for example, should be accessible by many agents in parallel, model space and other distributed aspects such as events that have a different dynamics from that of agents. Thus we believe it should be devised as a distinct part of the model.

Global and Local view. In CAS models, the agent behaviour could be specified individually (*local* view) and replicated in populations. At the same time, one can be interested in specifying the behaviour of groups of agents (*global* view), requiring that individuals adhere to it. So we need a language supporting the definition of the *micro-level* and its quantitative behaviour and at the same time supporting the specification of the *macro-level* non-functional (emergent) properties, which are related to availability, reliability, robustness, and performance. This is also related to the choice between centralised, decentralised, or mixed control.

Environment/Space Modelling and Awareness. A uniform language supporting various models of space and locality is highly desirable. Also, space awareness is an important feature for allowing the design of more sophisticated and realistic models. For example, communication within a range is a realistic assumption requiring space-awareness. Different representations can also affect the efficiency and applicability of analysis techniques, so the language should support explicit tuning of this aspect.

Bottom-up Design. The specification language should support incremental, compositional, and bottom-up design of the model by first specifying the *individual behaviour* of agents, then their *interaction mechanism*, and finally interaction with the *environment*.

Control over Abstraction Level. Collective systems modelling can easily give rise to systems that have an unmanageable size. The ability to move between different level of detail of the description, without the need to completely redesign the model, is an important and desired feature.

Modelling Adaptation and Goal-orientedness. Without an optimal behaviour for agents, it is common in Collective Systems to design adaptation patterns, allowing the agents to react to unpredicted environment conditions in order to achieve their goal. Modelling adaptation requires the ability to express agents behaviour in a goal-oriented fashion and to identify which events the agent is able to control [5].

Support for Design and Analysis Techniques. Different analysis techniques may require more or less strict conditions on the model under consideration. For example, statistical model checking allows the analysis of models of moderate size without requiring very strict conditions. On the other hand, mean-field/fluid-flow approximations techniques have stronger requirements but scale to much larger models. This trade-off between precision and ability to use analysis techniques of interest should be easily manageable in the language. In particular, satisfaction of the conditions for applicability of analysis techniques should not be left to the user to prove. Therefore, a language for modelling CAS should support different semantics, from a more detailed to a more abstract one, and provide parameters as well as syntactic restrictions that ensure the applicability of the techniques of interest.

2.3 A Running Example

The city bike sharing case study will be used in the rest of the report to provide evidence of usability of the different formalisms we shall introduce in the coming sections.

We consider a model of a *bike sharing service*, where we assume a city with m *parking stations*, each one with its *location* $\ell_i \in \mathit{Loc} = \{\ell_1, \ldots, \ell_m\}$, a number of *available bikes* b_i, and a number of *available parking slots* s_i (for $i = 1, \ldots, m$). Parking stations are in one-to-one correspondence with the set of possible locations, which should be considered as (disjoint) areas of influence in the city. We also assume that we have n *users* of the bike sharing service: at any time, each user is positioned in *one* location and can be in one of the two states *Pedestrian* and *Biker*. In each of those states, the user can move around the city (with speed depending on the state) according to preferences modelled by two *probability* transition matrices $Q_{\mathtt{b}}$ and $Q_{\mathtt{p}}$ of size $m \times m$ for the biker and the pedestrian state, respectively. The user becomes a *Biker* or a *Pedestrian* using transitions named *borrow* and *return*.

In our model we identify two populations: parking stations (containing parking-slots and bikes) and users (that may or may not have a bike).We would like to model bike borrowing/return from/to stations in a distributed and adaptive way (avoiding a central server being aware of bikes/parking-slots availability for each parking). A user in a location ℓ can borrow (or return) a bike by issuing a request (e.g. by means of a mobile phone application) to the bike sharing system. This request can be satisfied by a bike (or parking-slot) available within a neighbourhood of ℓ. The latter condition is specified via a neighbourhood predicate $\varphi_n(\ell, \ell')$ (modelling, for example, that a parking station ℓ' is easily reachable from ℓ, because it is near or may be reached by public transport etc.). The bike sharing system answers with the location of a parking station which has an available bike (or an available parking-slot), within a neighbourhood of ℓ. In this response there is a degree of flexibility provided by the condition $\varphi_n(\ell, \ell')$. This flexibility in the choice allows some control on which parking station is selected among those that are in the neighbourhood of the current user location (including itself), and can be used to re-balance bike/parking-slot availability by redirecting users to parking stations that have more available bikes (or parking-slots)

In the next sections we introduce various language dialects currently under exploration in the project. In particular, we would like to discuss the design choices and illustrate in practice the possible advantages and disadvantages. Language features are shown at work on this example and also discussed with respect to the analysis of Section 2.

3 StocS: A stochastic variant of SCEL

StocS is a *modelling and programming language* based on SCEL. SCEL (Software Component Ensemble Language) [9, 8], is a kernel language that takes a holistic approach to programming autonomic computing systems and aims at providing programmers with a complete set of linguistic abstractions for programming the behaviour of Autonomic Components (ACs) and the formation of Autonomic Component Ensembles (ACEs), and for controlling the interaction among different ACs. A SCEL program is formed by a set of components of the form $I\,[\,K,\,P]$. Each component consists of:

- a *knowledge repository* K, modelling the actual knowledge of the components;
- an interface I, providing named functionalities and other attributes;
- a *process* P, describing the actual behaviour of the component.

The *knowledge repository* K manages both *application data* and *awareness data*. Application data are used for enabling the progress of ACs' computations, while awareness data provide information about the environment in which ACs are running (e.g. monitored data from sensors) or about the status of an AC (e.g. its current location). The definition of SCEL abstracts away from a specific implementation of knowledge repository. It is only assumed that there are specific operations for adding *knowledge items* t to a repository ($K \oplus t$), for removing elements, selected by a template T, from a repository ($K \ominus T$), and for inferring elements from a repository ($K \vdash T$). In the sequel we let $\mathbb{V}$ denote the set of values, $\mathbb{K}$ denote the set of possible knowledge repositories (or equivalently knowledge states), $\mathbb{I}$ denote the set of knowledge items, $\mathbb{T}$ denote the set of knowledge templates. The latter are used to retrieve data from the knowledge repository.

The component *interface* I is used to publish structural and behavioral information about the component in the form of *attributes*, i.e. names acting as references to information stored in the component's knowledge repository. Let $\mathbb{A}$ be the set of attribute names (which include the constant id used to indicate the component identifier); an interface I is a function in the set $\mathbb{K} \to (\mathbb{A} \to \mathbb{V})$. An interface defines a (partial) function from a a knowledge-base, attribute-name pair to the domain of values. Among the possible attributes, id is mandatory and is bound to the name of the component. Component names are not required to be unique, so that replicated service components can be modelled. The *evaluation* of an interface I in a knowledge state K is denoted as $I(K)$. The set of possible interface evaluations is denoted by $\mathbb{E}$.

A *process* P, together with a set of process definitions, can be dynamically activated. Some of the processes in P execute local computations, while others may coordinate interaction with the knowledge repository or perform adaptation and reconfiguration. *Interaction* is obtained by allowing processes executed at a give AC to access knowledge in the repositories of other ACs. Processes can perform three different kinds of Actions: $\mathbf{get}(T)@c$, $\mathbf{qry}(T)@c$ and $\mathbf{put}(t)@c$, used to act over shared knowledge repositories by, respectively, withdrawing, retrieving, and adding information items from/to the knowledge repository identified by c. We restrict targets c to the distinguished variable self, that is used by processes to refer to the component hosting it, and to component *predicates* p, i.e. formulas on component attributes. A component $I\,[\,K,\,P]$ is *identified* by a predicate p if $I(K) \models \mathsf{p}$, that is, the interpretation defined by the evaluation of I in the knowledge state K is a model of the formula p. Note that here we are assuming a fixed interpretation for functions and predicate symbols that are not within the attributes ($\mathbb{A}$). E.g. $battery < 3$ is a possible predicate, where $<$ and 3 have a fixed interpretation, while the value of *battery* depends on the specific component addressed.

StocS extends SCEL by modelling the time duration of actions and by replacing non-determinism in the interaction with (local/remote) knowledge bases by probability distributions. StocS has a Continuous Time Markov Chains operational semantics. Enriching SCEL with information about action durations and adapting it to the modelling and programming of Collective Adaptive Systems poses several challenges which we summarise in this section. In particular, we discuss design choices

Systems:	Components:	Processes:	
$S ::= C \mid S \parallel S$	$C ::= I\,[K,\, P]$	$P ::= \mathbf{nil} \mid a.P \mid P + P \mid P \mid P \mid X \mid A(\bar{p})$	
Actions:		Targets:	Ensemble Predicates:
$a ::= \mathbf{get}(T)@c \mid \mathbf{qry}(T)@c \mid \mathbf{put}(t)@c$		$c ::= \mathsf{self} \mid \mathsf{p}$	$\mathsf{p} ::= tt \mid e \bowtie e \mid \neg\mathsf{p} \mid \mathsf{p} \wedge \mathsf{p}$

Table 1: StocS syntax (Knowledge K, Templates T, and Items t are parameters)

both taking into account the analysis of Section 2 and the need for effective approaches to manage model *complexity* and *size*.

3.1 Language Features

The syntax of StocS is presented in Table 1, where the syntactic categories of the language are defined. Systems are sets of Components in parallel. Components contain a process P, used to specify their *behaviour*, and a knowledge base K, used to model its internal *awareness state* as well as its *contact with the environment*. Selected features of the component, depending on its internal state, are made available to the external world through the *interface* I. Actions operate on *local* or *remote* knowledge-bases and have a Target to determine which other components are involved in the action, based on the values exposed by their interface and evaluated with respect to a Predicate.

Interaction with local or remote knowledge-bases is performed by sending/retrieving concrete knowledge items (elements of $\mathbb{I}$) as well as patterns, called knowledge templates (elements of $\mathbb{T}$). $\mathbb{K}$ is the set of possible knowledge states. So, in Table 1, $K \in \mathbb{K}$, $t \in \mathbb{I}$, and $T \in \mathbb{T}$.

Knowledge repository. Knowledge is a key feature of StocS because it allows us to model awareness, adaptation, and the environment in which the component exists. Each *knowledge repository* is completely described by a tuple $(\mathbb{K}, \mathbb{I}, \mathbb{T}, \oplus, \ominus, \vdash)$. The operators $\oplus$, $\ominus$, $\vdash$ are used to add, withdraw, and infer knowledge items to/from knowledge repositories in $\mathbb{K}$, respectively. In SCEL operations on knowledge induce a *nondeterministic* behaviour related to the selection of possible multiple knowledge items matching a template. In StocS a knowledge repository reacts *probabilistically* to the different operations. Indeed, $\oplus$, $\ominus$, $\vdash$ operators may modify the state of the knowledge and have the following signature, where $\mathsf{Dist}(X)$ denotes the class of probability distributions on a set X with finite support:

$$\oplus : \mathbb{K} \times \mathbb{I} \to \mathsf{Dist}(\mathbb{K}), \quad \ominus : \mathbb{K} \times \mathbb{T} \hookrightarrow \mathsf{Dist}(\mathbb{K} \times \mathbb{I}), \quad \vdash : \mathbb{K} \times \mathbb{T} \hookrightarrow \mathsf{Dist}(\mathbb{I}).$$

Function $\oplus$ is *total* and defines how a knowledge item can be inserted into a knowledge repository: $K \oplus t = \pi$ is the probability distribution over knowledge states obtained as the effect of adding t. Functions $\ominus$ and $\vdash$ are *partial* and compute the result of withdrawing (resp. inferring) a tuple matching a given template from a knowledge state. For instance, the outcome of $K \ominus T$ is a probability distribution over all pairs $(K, t) \in (\mathbb{K} \times \mathbb{I})$ such that the item t matches the template T. Intuitively, if $K \ominus T = \pi$ and $\pi(K', t) = p$ then, when one tries to remove an item matching template T from K, with probability p item t is obtained and the resulting knowledge state is K'. Function $\vdash$ is similar to $\ominus$ but it does not change the state of the knowledge.

3.2 Informal Semantics

The semantics of SCEL does not consider time related aspects of computation. More specifically, the execution of an action of the form $\mathbf{act}(T)@c\,.\,P$ (for $\mathbf{put}/\mathbf{get}/\mathbf{qry}$ actions) is described by a *single* and *atomic* transition of the underlying SCEL semantics. This entails that, on action completion, the knowledge repositories of all components involved in the action execution have been modified accordingly, abstracting away: (1) when the execution of the action starts, (2) when the possible destination components are required to satisfy p, and (3) when the process executing the action resumes execution (i.e. it becomes P).

In StocS we address those time-related issues at different levels of abstraction. Depending on the degree of detail in modelling these aspects, we have four different semantics: *action-oriented* (ACT-OR), *interaction-oriented* (INT-OR), *network-oriented* (NET-OR), and *activity-oriented* (ACTIV-OR). These semantics model the stochastic behaviour of a StocS system at different levels of abstraction, facilitating the management of the complexity of the model according to the application of interest. In the remaining part of this section we briefly discuss these four variants and their motivations. A detailed description of the four semantics can be found in [17, 18].

1. **Action-oriented Semantics**. Under this semantics an action of the form $\mathbf{act}(T)@c$ (respectively **put**/**get**/**qry**) when enabled generates a *single* transition and has a state residence time that is computed via an *action rates function* and depends on the *source* component interface, the *target* component interface, and the retrieved knowledge item.

   This semantics does not consider all the realistic aspects of predicate-based communication like, for instance, the time needed to send a tuple item to *all the components* satisfying a given predicate. However, it can be used when actions average execution time does not depend on the number of components involved in the communication.

2. **Interaction-oriented Semantics**. This is based on the action-oriented semantics and distinguishes local and remote actions by assuming that local actions are executed instantaneously. This kind of semantics is useful when local actions happen in a time-scale which is very different (usually much smaller) from that of remote actions. In these situations it is reasonable to consider as instantaneous the execution of local actions. As a useful side effect of ignoring the duration of local actions, we obtain more concise models. This approximation can be considered as an approach to reducing multi-scale models to single-scale models. A similar idea is explored for Bio-PEPA models in [12] and used to abstract away from fast reactions in biochemical networks. The assumption we make for defining this semantics is that each remote (stochastically timed) action is followed by a (possibly empty) sequence of local (probabilistic) actions. We ensure this assumption is satisfied by imposing syntactic restrictions on processes. Then, by realising a form of maximal progress [14] we execute a timed action and all of its subsequent probabilistic actions in a single transition.

3. **Network-oriented Semantics**. This semantics provides the most detailed modelling among the four semantics, which entails that actions are *non-atomic*. Indeed, they are executed through several intermediate steps, each of which requires appropriate time duration modelling. In particular, **put** actions are realised in two steps: (*i*) an envelope preparation and shipping (one for each component in the system, other than the source), (*ii*) envelope delivery, with its own delivery time, test of the truth value of the communication predicate, and update of the knowledge-state. Also the actions **get**/**qry** are realised in two steps: (*i*) initiation of the item retrieval by a source component by entering in a waiting state, (*ii*) synchronisation with a destination component and exchange of the retrieved item. Since actions are not executed atomically, their (partial) execution is interleaved with that of other actions executed in parallel.

4. **Activity-oriented Semantics**. This semantics is very abstract and allows us to explicitly declare as atomic an entire sequence of actions, by assigning to it an execution rate of the entire sequence. Since the execution of the sequence of actions is atomic, it allows no interleaving of other actions. As an interesting consequence of this, we have a significant reduction in the state-space of the system. This variant of the semantics is motivated by the fact that sometime it is useful to perform a sequence of actions to update the knowledge repositories on different components. The Activity-oriented semantics allows us to declare as atomic an entire sequence of actions and to assign a rate to it. More generally, the purpose of this semantics is to have a very high-level abstraction of the interaction mechanisms.

$P_u \triangleq \mathit{Pedestrian}$

$$
\begin{array}{rlrl}
\mathit{Pedestrian} \triangleq & \mathbf{get}(\mathtt{p\_next}, L)@\mathsf{self}. & \mathit{Biker} \triangleq & \mathbf{get}(\mathtt{b\_next}, L)@\mathsf{self}. \\
 & \mathit{Borrow} & & \mathit{Return} \\
 & & & \\
\mathit{Borrow} \triangleq & \mathbf{qry}(\mathtt{loc}, L)@\mathsf{self}. & \mathit{Return} \triangleq & \mathbf{qry}(\mathtt{loc}, L)@\mathsf{self}. \\
 & \mathbf{get}(\mathtt{bike\_res}, \mathit{ID})@\mathsf{near}(L). & & \mathbf{get}(\mathtt{slot\_res}, \mathit{ID})@\mathsf{near}(L). \\
 & \mathbf{put}(\mathtt{go}, \mathit{ID})@\mathsf{self}. & & \mathbf{put}(\mathtt{go}, \mathit{ID})@\mathsf{self}. \\
 & \mathbf{get}(\mathtt{bike})@\mathsf{loc}(\mathit{ID}). & & \mathbf{put}(\mathtt{bike})@\mathsf{loc}(\mathit{ID}). \\
 & \mathbf{put}(\mathtt{b})@\mathsf{self}. & & \mathbf{put}(\mathtt{p})@\mathsf{self}. \\
 & \mathit{Biker} & & \mathit{Pedestrian}
\end{array}
$$

Figure 1: User behavior as a StocS process.

3.3 Example

Our design choice to model the bike sharing system response to bike/slot requests is to make parking stations themselves in charge of answering. This distributed solution is more efficient, robust, and compositional (i.e. the service can scale without changing the infrastructure) than a centralised one. A further advantage is that we can easily model a control over bike redistribution to minimise the cost of bike reallocation by means of trucks. In particular, the choice between parking stations is realised by using different response rates of parking stations: those that have more bikes available will have higher response rates to bike requests. Similarly, parking stations that have more slots available will have higher response rates to slot requests. By using a **get** action, we put these responses into a race and, on average, users will be redirected to those services that need bike or slot re-balancing. We will discuss this and other features of the model in the rest of the section.

A single user is represented as a component $I_u\,[\,K_u,\,P_u\,]$, whose knowledge state K_u is an element $\langle s, \ell\rangle$ in $\{\mathtt{b},\mathtt{p}\}\times \mathit{Loc}$ denoting the user state (i.e. either being a pedestrian or a biker) and the user location, and whose interface I_u, which defines the predicates biker, $\mathsf{pedestrian}$, and $\mathsf{loc}(\ell)$ as follows: $I_u(\langle \mathtt{b}, \ell\rangle) \models \mathsf{biker}$, $I_u(\langle \mathtt{p}, \ell\rangle) \models \mathsf{pedestrian}$, and $I_u(\langle s, \ell\rangle) \models \mathsf{loc}(\ell)$, for every $\ell \in \mathit{Loc}$ and $s \in \{\mathtt{b},\mathtt{p}\}$. Let us summarise the role of the user knowledge operators (see [18] for details). The $\oplus$ operator allows: to change state by $\oplus(\mathtt{b})$ (change to biker state) and by $\oplus(\mathtt{p})$ (change to pedestrian state), and to move to a specified location ℓ' by $\oplus(\mathtt{go}, \ell')$. The $\ominus$ operator allows movement to a location according to the average user behavior in the pedestrian state, by $\ominus(\mathtt{p\_next}, L)$, and in the biker state, by $\ominus(\mathtt{b\_next}, L)$. Finally, the $\vdash$ operator allows retrieval of the current user location.

The users behaviour is given in Figure 1. Each user starts in the state *Pedestrian*, where movement is possible through a local **get** of the item `p_next`. The effect of this action is to change user location into ℓ_j. The latter is also returned as a binding for the variable L. This information will be used to compute the rate of the action (i.e. of the movement) by a suitable rate function $\mathcal{R}$. For instance, the rate of action $\mathbf{put}(\mathtt{go}, \ell')@\mathsf{self}$ used to model the movement of a user towards location ℓ' is defined as follows[1]:

$$
\mathcal{R}(\sigma, \mathbf{put}(\mathtt{go}, \ell')@\mathsf{self}, \sigma) \quad = \quad \begin{cases} \lambda_{\mathtt{b}} \cdot f_{dist}(\ell, \ell') & \text{if } \sigma \models \mathsf{biker} \wedge \mathsf{loc}(\ell) \\ \lambda_{\mathtt{p}} \cdot f_{dist}(\ell, \ell') & \text{if } \sigma \models \mathsf{pedestrian} \wedge \mathsf{loc}(\ell) \end{cases}
$$

By using this function, the rate of a movement depends on the state of the user (pedestrian or biker) and on the distance between the current and the target location.

The process *Borrow* first retrieves the current location L then performs a *bike* reservation (`bike_res`) from a parking station ID satisfying predicate $\mathsf{near}(L)$. The actual rate of this action *depends on available bikes*: the higher the number of available bikes is, the higher the execution rate is. As an effect of this race condition, the ID of the near station containing *more bikes* is received by the user with a

[1]Due to space limitation, we leave out the definition of $\mathcal{R}$, which can be found in [18].

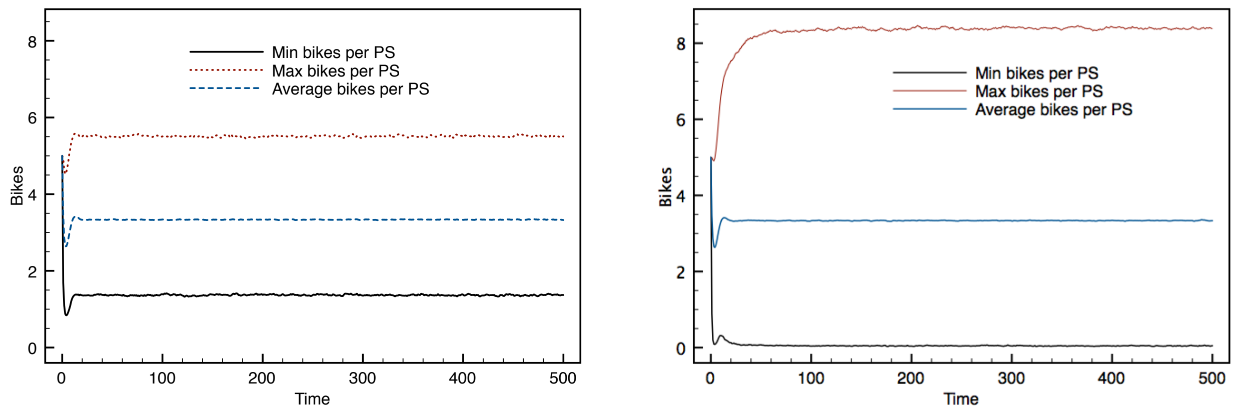


Figure 2: Simulation of bike sharing service.

higher probability than a near station with *fewer bikes*, causing a more balanced distribution of bikes in the system. When the parking-slot is reserved, the user moves towards the parking stations. The rate of this action depends on the distance between the user and the parking station. After that process *Borrow* takes a `bike`; this operation is performed via a **get** action that decrements the bikes available and increments the slots available. Finally, the user status is updated to biker `b`. A biker moves around the city and, then, leaves his bike in a parking station by executing the process *Return*. Its behaviour is similar to that of a pedestrian, except for the fact it reserves *parking slots* instead of bikes.

A parking station is represented as a component $I_p\,[K_p,\, \mathbf{nil}]$ that has no behaviour (it is passive). Its knowledge state is a vector $\langle b_a, b_r, s_a, s_r, \ell\rangle \in \mathbb{N}^4 \times \mathit{Loc}$ denoting the number of available bikes (b_a), of reserved bikes (b_r), of available parking slots (s_a), and of reserved parking slots (s_r), as well as the parking location ℓ. The parking station interface I_p defines the predicates $\mathsf{loc}(\ell)$ and $\mathsf{near}(\ell)$ as follows [18]: for every $\ell, \ell' \in \mathit{Loc}$ and $b_a, b_r, s_a, s_r \in \mathbb{N}$ $I_p(\langle b_a, b_r, s_a, s_r, \ell\rangle) \models \mathsf{loc}(\ell)$ and $I_p(\langle b_a, b_r, s_a, s_r, \ell\rangle) \models \mathsf{near}(\ell')$ if $\varphi_n(\ell, \ell')$, where $\varphi_n(\ell, \ell')$ is a suitable neighbourhood predicate. An initial state of this model is a term

$$\|_{i=1}^{m} \;\big(\; (I_u\,[\,\langle \ell_i, \mathsf{p}\rangle,\, P_u\,])[k_i] \,\|\, I_p\,[\,\langle b_i, 0, s_i, 0, \ell_i\rangle,\, \mathbf{nil}\,]\;\big)$$

which denotes, for $i = 1, \ldots, m$: (i) k_i pedestrians in locations ℓ_i, and (ii) b_i available bikes and s_i available parking slots in parking station at location ℓ_i. Note that the number of reserved bikes as well as the number of reserved slots is set to zero in the initial state of the system in every parking station. The overall number of bikes in the system is preserved by the knowledge-update rules. Some simple simulation analyses of the considered system are reported in Figure 2. These simulations, based on action oriented semantics, have been performed with jRESP[2]. This is a Java framework that can be used to execute and simulate SCEL/StocS specifications. Figure 2 compares the simulation results of the considered case study where rates of bike and slot reservations depend on the number of available resources (on the left) with the one where these rates are constant[3] (on the right). We can notice that the average number of available bikes in parking stations is similar in the two simulations. However, when the bikes/slots reservation rate depends on the available resources, the bikes are more evenly distributed over the different parking stations.

3.4 Discussion

We evaluate StocS against the features discussed in Section 2.

[2]`http://jresp.sourceforge.org`

[3]We consider 40 users moving over a grid of 4×4 locations. Each parking station starts with 5 bikes and 5 empty slots.

Automated Derivation of the Model. StocS has four stochastic semantics with different level of detail. Their formal semantics is already fully defined and can be used to derive automatically a CTMC model from a StocS specification.

Agents + Environment view. In StocS, agents are generally modelled through components, while the model of the environment is given by specifying the knowledge behaviour.

Global and Local view. Both the global and local views are addressed in StocS. Local view is modelled via components and agents behaviour while the global view emerges from the components interactions based on predicates.

Environment/Space Modelling and Awareness. In StocS the environment where components operate is modelled via specific data available in the *knowledge state*. This enables both environment awareness (agents behaviour can depend on the knowledge state) and, thanks to the predicate based interaction specific of StocS, component interactions can be regulated/effected by the current space configuration.

Bottom-up Design. The individual level is given by specifying the behaviour of single component. The interaction is determined by the choice of blocking/nonblocking and one-to-one/multi-cast actions as well as the chosen semantics (e.g. interaction-oriented abstracts local actions, while activity-oriented focuses on sequences). Finally, the interaction with the environment is managed both by the choice of the action performed and by the choice of the knowledge behaviour.

Control over Abstraction Level. StocS addresses this in two ways: (1) by assigning different stochastic semantics for actions to the same syntax, thus allowing management of the size of the underlying model without changing the design, (2) by exploiting the separation between the environment model and the agent behaviour, thus allowing consideration of different models of space and corresponding different interpretations for communication predicate (e.g. graph based vs continuous space, and corresponding neighbourhood function).

Modelling Adaptation and Goal-orientedness. By accessing and manipulating knowledge repositories (local or remote), StocS components acquire information about their status (self-awareness) and their environment (context-awareness) and can perform self-adaptation, initiate self-healing actions to deal with system malfunctions, or install self-optimising behaviours [8].

Support for Design and Analysis Techniques. Currently StocS does not provide any specific tools supporting design and formal analysis. However, jRESP and its simulation API can be used to simulate StocS specification as shown in the previous subsection.

4 PALOMA

PALOMA, the Process Algebra of Located Markovian Agents, is intended to provide a simple process algebra-based formalism which can be used to generate models in the style of M^2MAM [6]. M^2MAM is used to generate a mean field model, but being based on Markovian agents it is also amenable to a discrete interpretation. The mean field model is specified by a number of matrices and vectors, and can be analysed by solving a set of coupled ODEs. The intention is that PALOMA should be equipped with both a discrete and a continuous semantics. The discrete semantics provides theoretical foundations for discrete event simulation of the models. The continuous semantics facilitate the automatic derivation of the matrices, and thus the underlying vector field ODEs of a M^2MAM model, from the high-level description. Here we focus on the discrete interpretation.

4.1 Language Features

In PALOMA each agent is a finite state machine and the language is *conservative* in the sense that no agents are spawned or destroyed during the evolution of a model (although they can cease to change state). The language has a two level grammar:

$$\begin{aligned} X(\ell) &::= \ !(\alpha, r).X(\ell) \mid ?(\alpha, p).X(\ell) \mid X(\ell) + X(\ell) \\ P &::= \ X(\ell) \mid P \parallel P \end{aligned}$$

Agents are parameterised by a *location*, here denoted ℓ. Agents can undertake two types of actions, *spontaneous* actions, denoted $!(\alpha, r)$, and *induced* actions, denoted $?(\alpha, p)$. When an agent performs a spontaneous action, it does so with a given rate r, which is taken to be the parameter of an exponential distribution, where $1/r$ is the expected duration of the action. Spontaneous actions are broadcast to the entire system, and can induce change in any other agent which enables an induced action with the matching type α. An induced action has an associated probability p, which records the probability that the agent responds to a spontaneous action of the same type. In the style of the Calculus of Broadcasting Systems (CBS) [20], this can be thought of as the probability that the agent *hears* the propagated message. Alternative behaviours are represented by the standard choice operator, $+$. A choice between spontaneous actions is resolved via the race policy, based on their corresponding rates. We assume that there is never a choice between induced actions of the same type.

A model, P, consists of a number of agents composed in parallel. There is no direct communication between agents, for example in the style of shared actions in PEPA or CSP. Instead, all communication/interaction is via spontaneous/induced actions. When an action is induced in an agent the extent of its impact is specified by a *perception function*, $u(\alpha, \ell, X, \ell', X')$, where α is the type of the spontaneous action, ℓ and ℓ' are the locations of the sender and receiver respectively, and X and X' indicate that their current states are $!(\alpha, r).X$ and $?(\alpha, p).X'$ respectively for some r and p. This is a further probability which, given the locations of the two agents, their current states and action type involved, determines the likelihood that the induced action occurs. In the terminology of CBS, this is the probability that an agent who *hears* a message actually *listens*. This message passing modified by perception functions can be used to implement attribute-based communication. For example, the perception function might have value 1 when the two agents are within a communication radius r of each other, but a value of 0 whenever the distance between them is greater than r. Obviously this gives a rich set of possible styles of interaction, but note that each agent with an induced action chooses independently whether to respond or not.

4.2 Semantics

The semantics proceeds in sequences of alternating steps. This can be regarded as giving rise to a semi-Markov process in which the first step, corresponding to the spontaneous action, determines a delay, whilst the second step, is probabilistic and determines what the next state will be. This corresponds to each possible induced action making the choice of whether to respond, based both on its inherent probability of "hearing" and the perception function. As each agent makes such a decision independently, the probabilities can be multiplied to obtain the overall probability of a given next state.

As with all calculi with broadcast actions, there is a challenge to know when the broadcast is "complete" in some sense, i.e. when all possible agents have "heard" the message. We tackle this problem by associating an *ether* element with the system, which provides the environment for *all* agents. A spontaneous action can only be initiated if the ether is currently empty, E_0 and no probabilistic transitions are enabled. A spontaneous action is deemed to be complete when all agents have moved to a probabilistic state. At this point, a probabilistic resolution must be made to determine the next state. This will have the effect of clearing the ether and creating the opportunity for the next spontaneous message. In some cases a spontaneous action may not induce any actions in other agents. If this is the case the message will propagate, without impacting any other agents, and the probabilistic step will be trivially resolved.

$$E_0, !(\alpha,r).X(\ell) \xrightarrow{(\alpha,r)} [\alpha,r,\ell,X], X(\ell)^{\mathcal{P}} \qquad \not\rightarrow_{\mathcal{P}}$$

A spontaneous action can occur if and only if the ether is empty, denoted by E_0. The resulting local state records that the ether contains the message α which originated with rate r at location ℓ from the state $!(\alpha,r).X(\ell)$, and that the continuation is subject to a probabilistic resolution.

$$[\alpha,r,\ell',X'], ?(\alpha,p).X(\ell) \xrightarrow{(\alpha,r)} [\alpha,r,\ell',X'], (?(\alpha,p).X(\ell) +^p X(\ell))^{\mathcal{P}}$$

If the ether contains a message α then an agent that enables an induced action of that type may progress to a state subject to a probabilistic resolution in which, with probability p, the continuation $X(\ell)$ is chosen, and with probability $1-p$, the continuation $?(\alpha,p).X(\ell)$ is chosen (the agent does not "hear" to the message in the ether).

$$[\alpha,r,\ell',X'], !(\beta,s).X(\ell) \xrightarrow{(\alpha,r)} [\alpha,r,\ell',X'], !(\beta,s).X(\ell)^{\mathcal{P}}$$

If the ether contains a message of type α then an agent that enables a spontaneous message of any type (including α) witnesses the ongoing action, enters a probabilistic state and awaits resolution. This ensures that only one spontaneous action can be in progress at a time. Note that there is not the possibility of an agent "sharing" the α action as would be possible in a language such as CSP or PEPA.

$$[\alpha,r,\ell,X'], ?(\beta,p).X(\ell) \xrightarrow{(\alpha,r)} [\alpha,r,\ell,X'], ?(\beta,p).X(\ell)^{\mathcal{P}} \quad \beta \neq \alpha$$

Similarly, if the ether contains a message of type α then an agent that enables an induced message of any type except α witnesses the ongoing action, enters a probabilistic state and awaits resolution.

$$\frac{E, X_1(\ell) \xrightarrow{(\alpha,r)} E', X_1'(\ell')^{\mathcal{P}}}{E, X_1(\ell)+X_2(\ell) \xrightarrow{(\alpha,r)} E', X_1'(\ell')^{\mathcal{P}}} \qquad \frac{E, X_2(\ell) \xrightarrow{(\alpha,r)} E', X_2'(\ell')^{\mathcal{P}}}{E, X_1(\ell)+X_2(\ell) \xrightarrow{(\alpha,r)} E', X_2'(\ell')^{\mathcal{P}}}$$

Choice behaves as we would anticipate. We assume that within a choice both elements are in the same location as they correspond to a single agent.

$$\frac{E_1, X_1(\ell_1) \xrightarrow{(\alpha,r)} E', X_1'(\ell_1')^{\mathcal{P}} \qquad E_2, X_2(\ell_2) \xrightarrow{(\alpha,r)} E', X_2'(\ell_2')^{\mathcal{P}}}{(E_1, X_1(\ell_1)) \parallel (E_2, X_2(\ell_2)) \xrightarrow{(a,r)} E', (X_1'(\ell_1') \parallel X_2'(\ell_2'))^{\mathcal{P}}}$$

Parallel agents may progress in parallel if and only if they agree on the spontaneous action to take place, (α,r), and consequently update the ether in the same way.

The probabilistic resolutions are determined by a second transition relation $\longrightarrow_{\mathcal{P}}$, defined as follows. We assume that no spontaneous transitions, $\xrightarrow{(\alpha,r)}$, are enabled.

$$[\alpha,r,\ell',X'], (?(\alpha,p).X(\ell) +^p X(\ell))^{\mathcal{P}} \begin{cases} \xrightarrow{(\alpha,p\times u(\alpha,\ell',X',\ell,X))}_{\mathcal{P}} & E_0, X(\ell) \\ \xrightarrow{(\alpha,1-(p\times u(\alpha,\ell',X',\ell,X)))}_{\mathcal{P}} & E_0, ?(\alpha,p).X(\ell) \end{cases}$$

The only probabilistic states which genuinely have different possible outcomes are those which resulted from an induced action. In this case there are two different resolutions according to whether the induced action is "heard" or not. In either case the ether is emptied when the probabilistic resolution is made. First, a choice is made whether to hear the message or not, but secondly, if the message is heard, its impact is adjusted according whether it is listened to as governed by the perception function. This is in keeping with the M^{2}MAM framework [6].

For other states the probabilistic resolution will simply clear the ether and return the agent to an active state again.

$$[\alpha, r, \ell', X'], X(\ell)^{\mathcal{P}} \xrightarrow{(\alpha,1)}_{\mathcal{P}} E_0, X(\ell)$$

For parallel agents, they undergo probabilistic resolution independently and their probabilities are multiplied.

$$\frac{E, X_1(\ell_1)^{\mathcal{P}} \xrightarrow{(\alpha,p)}_{\mathcal{P}} E_0, X_1'(\ell_1') \qquad E, X_2(\ell_2)^{\mathcal{P}} \xrightarrow{(\alpha,q)}_{\mathcal{P}} E_0, X_2'(\ell_2')}{E, (X_1(\ell_1) \parallel X_2(\ell_2))^{\mathcal{P}} \xrightarrow{(\alpha,p\times q)} (E_0, X_1'(\ell_1')) \parallel (E_0, X_2'(\ell_2'))}$$

4.3 Example

Here we present a PALOMA model of the bike-sharing system described in Section 3. Recall that there are m bike stations in the city, and each one has a number of available bikes and slots. Therefore, we represent the available bikes and slots in Station i (for $i = 1, ..., m$) by agents in the following two states:

$$\begin{aligned} \mathit{Slot}(\ell_i) &= ?(\mathit{return}, 1).\mathit{Bike}(\ell_i) \\ \mathit{Bike}(\ell_i) &= ?(\mathit{borrow}, 1).\mathit{Slot}(\ell_i) \end{aligned}$$

Both $\mathit{Slot}(\ell_i)$ and $\mathit{Bike}(\ell_i)$ are passive. They can only be induced to make a *return* (a bike is returned to this station) or *borrow* (a bike is borrowed from this station) action, and when they do they switch to each other.

We introduce a three-state agent to represent the bike station, which is defined as:

$$\begin{aligned} \mathit{EmptyStation}(\ell_i) &= ?(\mathit{return}, 1).\mathit{NormalStation}(\ell_i) + !(\mathit{SlotAvailable}, \sigma).\mathit{EmptyStation}(\ell_i) \\ \mathit{NormalStation}(\ell_i) &= ?(\mathit{return}, 1).\mathit{FullStation}(\ell_i) + ?(\mathit{borrow}, 1).\mathit{EmptyStation}(\ell_i) \\ &\quad + !(\mathit{SlotAvailable}, \sigma).\mathit{NormalStation}(\ell_i) + !(\mathit{BikeAvailable}, \sigma).\mathit{NormalStation}(\ell_i) \\ \mathit{FullStation}(\ell_i) &= ?(\mathit{borrow}, 1).\mathit{NormalStation}(\ell_i) + !(\mathit{BikeAvailable}, \sigma).\mathit{FullStation}(\ell_i) \end{aligned}$$

The $\mathit{EmptyStation}(\ell_i)$ state denotes that there are no available bikes in Station i, whereas the $\mathit{FullStation}(\ell_i)$ state denotes that there are no available slots. When the bike station agent is in the *NormalStation* state, it can broadcast both *BikeAvailable* and *SlotAvailable* messages at the rate of σ. However, when it is in the *EmptyStation* state, it can only broadcast *SlotAvailable* messages. Similarly, it can only broadcast *BikeAvailable* messages when it is in the *FullStation* state. Moreover, an agent in the *EmptyStation* state can receive a *return* message and go back to the *NormalStation* state. Similarly, an agent in the *FullStation* can go back to the *NormalStation* state by accepting a *borrow* message. The agent in the *NormalStation* state can also act on *return* or *borrow* messages, leading to the *FullStation* state or the *EmptyStation* state respectively.

The perception functions for the messages *borrow* and *return* are defined as follows:

$$u(\mathit{borrow},\ell,X,\ell',X') = \begin{cases} \dfrac{1}{b_\ell+\delta} & \text{if } (\ell=\ell' \wedge X=\mathit{Bike}) \\ \dfrac{|2-b_\ell|+2-b_\ell}{4-2\times b_\ell+\delta} & \text{if } (\ell=\ell' \wedge X=\mathit{NormalStation}) \\ 1 & \text{if } (\ell=\ell' \wedge X=\mathit{FullStation}) \\ 0 & \text{otherwise} \end{cases}$$

$$u(\mathit{return},\ell,X,\ell',X') = \begin{cases} \dfrac{1}{s_\ell+\delta} & \text{if } (\ell=\ell' \wedge X=\mathit{Slot}) \\ \dfrac{|2-s_\ell|+2-s_\ell}{4-2\times s_\ell+\delta} & \text{if } (\ell=\ell' \wedge X=\mathit{NormalStation}) \\ 1 & \text{if } (\ell=\ell' \wedge X=\mathit{EmptyStation}) \\ 0 & \text{otherwise} \end{cases}$$

where δ is a tiny positive real number to ensure that the denominator is never equal to 0. b_ℓ and s_ℓ represent the number of agents in *Bike* and *Slot* states in location ℓ respectively, as explained in Section 2 . The perception function of the *borrow* message can be explained in the following way:

- If the message is received by a *Bike* agent in the same location as the sender, it can be percieved with probability $1/(b_\ell+\delta)$. This means that probabilistically only one bike is borrowed for each message broadcast.
- If the message receiver is the *NormalStation* agent in the same location as the message sender, it can be perceived with probability $\dfrac{|2-b_\ell|+2-b_\ell}{4-2\times b_\ell+\delta}$, which works as a guard to ensure the message can only be perceived when there is only one available bike in the station.
- If the message receiver is the *FullStation* agent in the same location as the message sender, it must be percieved.
- Otherwise, the message cannot be perceived by the receiving agent.

The perception function of *return* message is analogous.

The agents representing bike users have six states, which are:

$$\begin{aligned} \mathit{Pedestrian}(\ell_i) &= !(\mathit{seekb}_i,t).\mathit{SeekBike}(\ell_i) + \sum_{j\neq i} !(\mathit{walk}_{ij},Q_{\mathtt{P}}(i,j)).\mathit{Pedestrian}(\ell_j) \\ \mathit{SeekBike}(\ell_i) &= ?(\mathit{BikeAvailable},1).\mathit{BorrowBike}(\ell_i) + \sum_{j\neq i} !(\mathit{seekb}_j,w_{ij}).\mathit{SeekBike}(\ell_j) \\ \mathit{BorrowBike}(\ell_i) &= !(\mathit{borrow},o).\mathit{Biker}(\ell_i) \\ \mathit{Biker}(\ell_i) &= !(\mathit{seeks}_{,i}\,r).\mathit{SeekSlot}(\ell_i) + \sum_{j\neq i} !(\mathit{ride}_{ij},Q_{\mathtt{b}}(i,j)).\mathit{Biker}(\ell_j) \\ \mathit{SeekSlot}(\ell_i) &= ?(\mathit{SlotAvailable},1).\mathit{ReturnBike}(\ell_i) + \sum_{j\neq i} !(\mathit{seeks}_j,c_{ij}).\mathit{SeekSlot}(\ell_j) \\ \mathit{ReturnBike}(\ell_i) &= !(\mathit{return},o).\mathit{Pedestrian}(\ell_i) \end{aligned}$$

As can be seen from the definition, when the user agent is in the *Pedestrian* state, it travels from location ℓ_i to location ℓ_j at the rate of $Q_{\mathtt{P}}(i,j)$. This spontaneous action does not have any corresponding induced actions and so can be thought of as not emitting a message. It may also seek a bike at the rate of t, and goes into the *SeekBike* state (the number of users in this state at location ℓ is denoted sb_ℓ). The user agent in the *SeekBike* state can accept a *BikeAvailable* message and go to the

BorrowBike state. Alternatively, it can also move to other locations and seek an available bike in that station with rate w_{ij}. The borrow bike action is fired at rate o, a message *borrow* is emitted, and the user becomes a *Biker* at the same location. A user agent in the *Biker* state in location ℓ_i travels to location ℓ_j at the rate of $Q_{\mathtt{b}}(i, j)$. It may also seek a slot at the rate of r, and goes into the *SeekSlot* state(the number of users in this state at location ℓ is denoted ss_ℓ). The user agent in the *SeekSlot* state may be induced in a *SlotAvailable* action by a station agent and go to the *ReturnBike* state. The return bike action is also fired at rate o. Meanwhile, a message *return* is emitted during the action, and the user becomes a *Pedestrian* again after the transition.

The definition of the perception functions for messages *BikeAvailable* and *SlotAvailable* are given as follows:

$$u(BikeAvailable, \ell, X, \ell', X') = \begin{cases} \dfrac{1}{sb_\ell + \delta} \times \dfrac{b_\ell}{b_\ell + s_\ell} & \text{if } (\ell = \ell' \wedge X = SeekBike) \\ 0 & \text{otherwise} \end{cases}$$

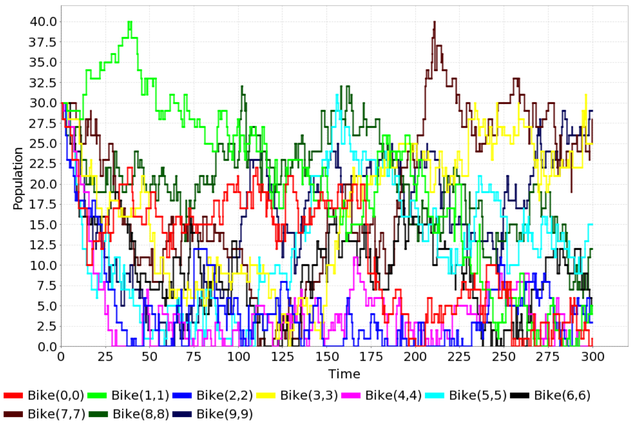
$$u(SlotAvailable, \ell, X, \ell', X') = \begin{cases} \dfrac{1}{ss_\ell + \delta} \times \dfrac{s_\ell}{b_\ell + s_\ell} & \text{if } (\ell = \ell' \wedge X = SeekSlot) \\ 0 & \text{otherwise} \end{cases}$$

where the terms $\frac{1}{sb_\ell+\delta}$ and $\frac{1}{ss_\ell+\delta}$ (sb_ℓ and ss_ℓ denote the number of agents in *SeekBike* and *SeekSlot* states in location ℓ respectively) ensure that bikes are borrowed or returned probabilistically one by one. Moreover, the terms $\frac{b_\ell}{b_\ell+s_\ell}$ and $\frac{s_\ell}{b_\ell+s_\ell}$ represent the incentive factor, which makes it less possible for the users to percieve the messages from the stations with fewer available bikes or slots. Thus, they are more likely to seek available bikes or slots in other stations.

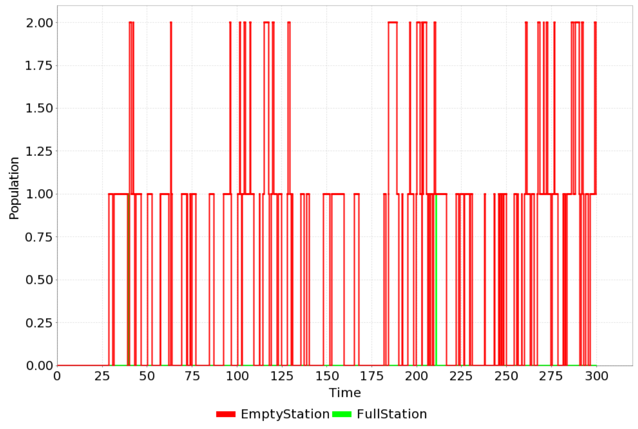
Finally, the initial population of agents are given in the following definition:

$$\ldots \parallel Pedestrain(\ell_i)[NP_i] \parallel Slot(\ell_i)[NS_i] \parallel Bike(\ell_i)[NB_i] \parallel Station(\ell_i) \parallel \ldots$$

We are able to analyse the model both as a discrete event simulation and through the mean field ODEs with software developed during the QUANTICOL project. Figure 3 and 4 illustrates the trajectories of number of available bikes in stations and number of problematic stations (empty and full stations) in our simulation with and without the incentive factor respectively, in which $NP_i = 50$, $NS_i = 10$, $NB_i = 30$ for $i = (0, \ldots, 9)$. It can be seen that the incentive factor makes a big difference in reducing the number of problematic stations.

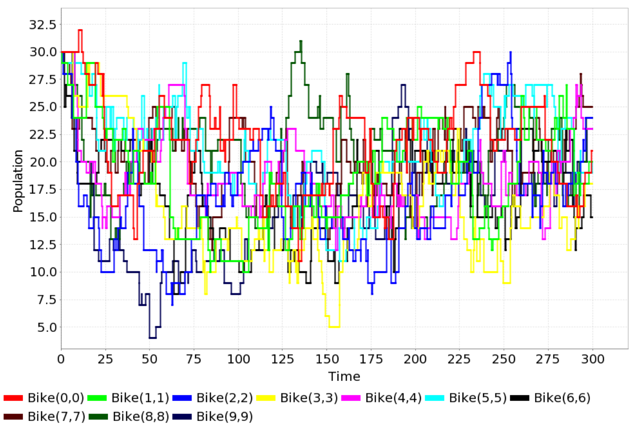


(a) The trajectories of number of available bikes in stations

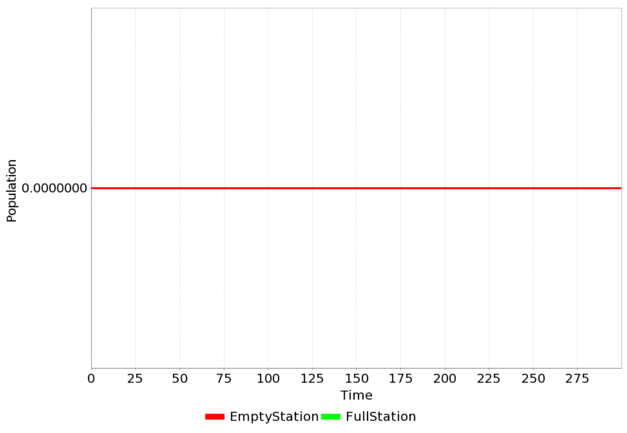


(b) The trajectories of number of problematic stations

Figure 3: Plots without the incentive factor



(a) The trajectories of number of available bikes in stations



(b) The trajectories of number of problematic stations

Figure 4: Plots with the incentive factor

4.4 Discussion

We evaluate PALOMA against the features discussed in Section 2.

Automated Derivation of the Model. The language is supported by software tools to automatically derive both a discrete event simulation (as in the plots shown in Figures3 and 4, and mean-field approximations as derived for M^2MAM).

Agents + Environment view. The agents in the system are captured by agents in the process algebra, whilst the environment is captured more implicitly in the locations, the perception functions and counts of the number of agents in each possible state.

Global and Local view. The specification of each agent with a given location provides a mechanism to develop a representation of local behaviour, whilst the global behaviour is more emergent, since it results from the evolution of the agents according to their local behaviours and the influence of the perception functions.

Environment/Space Modeling and Awareness. The current version of the language is based on a static notion of environment as the perception functions depend only on the current state of the agents and locations which do not change during the execution of the model. However it would be possible to add further dependence into the perception function, for example capturing a radius of perception that changes over time. But this would be somewhat at odds with the design of the language which is intended to be minimal for exploration of the particular issue of attribute defined communication.

Bottom-up Design. As with all process algebras, it is natural to build a model by first defining the capabilities of the individual agents and then specifying how they interact. Thus the logical behaviour of the entities in the system is readily designed in a bottom-up style. But the communication between agents is controlled by the perception function which may require a more global view of the system.

Control over Abstraction Level. Of course the modeller has a choice of what to choose as the agents within the system, or how finely to distinguish locations, but once this choice has been made PALOMA does not currently support any automated approaches to defining more abstract views of the system. This is an area for future investigation.

Modeling Adaptation and Goal-orientedness. The perception functions can be used to give quite rich (global) state dependences on the behaviour of the agents, particularly in their response to the spontaneous actions of other agents. This can be viewed as a form of adaptation. But it is rather restricted and further mechanisms could be investigated in the future. There are no structures within the language to capture goals. For this language it would be more appropriate to specify goals

$$S ::= !(\alpha, r).X \mid ?(\alpha, w).X \mid !!(\alpha, r).X \mid ??(\alpha, w).X \mid S + S$$
$$X \overset{def}{=} S \qquad D ::= X \mid D \parallel D$$
$$P ::= a\{D\} \mid P \underset{L}{\bowtie} P \qquad W ::= [P](\ell) \mid W \| W$$

Table 2: Syntax of PEPA-S.

in terms of a companion logic and then check the capacity of an agent to satisfy its goals.

Support for Design and Analysis Techniques. Our simulation tool can parse PALOMA models and run discrete event simulations. The corresponding mean-field model is automatically generated in the form of Matlab scripts when the model is parsed, and can be run directly in Matlab. Developing a front-end and graphical user interface for this tool is a priority for ongoing work.

5 PEPA-S

PEPA-S is an *ongoing* attempt to extend the stochastic process algebra PEPA [15] in a spatial context, adopting the spatial interaction mechanism developed in [6] for Markovian agent models. The choice of PEPA as the core language to describe agents is not a fundamental requirement: we believe similar ideas for a spatial extension can be applied to other process algebras. PEPA-S is a population language: we consider heterogeneous populations of indistinguishable agents. PEPA-S revolves around a few central ideas:

- Processes are located in a discrete set of spatial locations. In each location we have a population of interacting agents, so that the obtained class of models is a Patch PCTMC (see Deliverable 2.1).
- The spatial interactions are mediated via a broadcast-based communication. We also consider a handshake communication between agents in different locations (unicast).
- Actions are partitioned into actives and passives. Active actions are standard stochastic events happening locally (i.e. in a certain location) according to classical PEPA rules. However, active actions can trigger a broadcast or a unicast message through space. The response to those messages is carried out by passive actions.
- Passive actions respond to active messages with a certain probability, given by a so-called *perception function*. This function depends on the type of actions involved and on the locations of the sending agent (i.e. the location where the triggering active action happened), and of the receiving agent. Essentially, all the information about the spatial connectivity is encoded in such a function.

5.1 Language Features

The syntax of the language is given in Table 2. Sequential components, defined by $X \overset{def}{=} S$, are essentially simple automata (with a component name attached to each state) that can do four kinds of actions:

- **Active broadcast** $!(\alpha, r)$. This is an active action, hence happening at a rate r following standard PEPA rules. When executed, it triggers a broadcast.
- **Passive broadcast** actions $?(\alpha, w)$. These respond to an active action happening in the same or in a different location. w here is a weight which is used to probabilistically solve non-determinism due to the choice operator between two or more passive actions that can respond to a broadcast of α.

- **Active unicast** $!!(\alpha, r)$. These actions also behave like standard PEPA actions, with a rate r.
- **Passive unicast** $??(\alpha, w)$. These respond to an active unicast in a different location in the network. Only one passive unicast synchronises with an active unicast, defining a *handshake synchronisation*. The weight w is used to solve probabilistically a competition between two or more passive unicasts ready to reply to an active action.

Sequential agents (of the same agent class) are composed non-cooperatively in parallel (operator $\|$). A population $D = X_1 \parallel \ldots$ is then wrapped by the group operator $a\{D\}$, borrowed from [13], assigning it the name a. In specifying a population D, we use the shorthand $X_i[n]$ to indicate n copies in parallel of X_i: $X_i \parallel \ldots \parallel X_i$.

Populations of agents are then synchronised with the classical PEPA cooperation operator $\underset{L}{\bowtie}$, to define a population model P, which is then positioned in location v by the construct $[P](\ell)$. Spatial parallel composition $W \| \lfloor W$ then allows us to construct the spatial interaction network.

The syntax model is accompanied by another crucial piece of information, encoded in the *perception function* u, which specifies the spatial structure of interactions. The perception function, as defined in the current version of the language, takes four arguments, $u = u(\alpha, \ell, \beta, \ell')$, and returns the *probability* with which a passive action β responds to the active one α. It depends on the active action α that triggers a broadcast in location ℓ and on the passive action β that responds to the active one in location ℓ'. The dependence on locations allows us to encode the spatial structure in a very flexible way. We can model long range or short range interactions, allowing messages to propagate in the whole space or just to neighbouring locations, according to some notion of proximity between nodes. Furthermore, we do not require name matching, but adopt a more flexible key-hole interaction scheme, like in Blenx [21].

5.2 Informal Semantics

In order to describe the population model, we represent the state of the system according to the counting abstraction [16], introducing a vector of integers $\vec{\xi}$ which counts, for each location ℓ, each population a and each local state i of a the number of agents $\xi^{\ell}_{a,i}$ present in the system.

The semantics is then given similarly to [22]. We construct the reduced model, replacing a population $\ell\{D\}$ with the reduced population $a\{X_1 \parallel \ldots \parallel X_k\}$, where $X_1, \ldots, X_k$ are the different local states of population a. Then, we define a structural operational semantics which constructs a LTS with a single state, and many looping transitions, which encode all the possible behaviours of the model. In fact, transitions are labelled with a rate function, depending on the population vector $\vec{\xi}$, and with an update vector $\vec{\nu}$ giving the net change in each population due to the happening of the transitions.

Each of these transitions is composed by first deriving a rule for the active action, and then combining it with the rules of one (in case of unicast) or more (in case of broadcast) passive actions responding to it. There are several semantic relations, taking into account the different layers of the language and the different kind of spatial communications.

From this semantics, we can derive either a Population CTMC or a set of mean field equations. To construct an operational semantics capable of dealing with mean field approximation of broadcast in a consistent way, we encode the probabilistic response of passive actions due to the perception function using random update vectors. Essentially, instead of considering all the possible combinations of agents that respond and do not respond to an active message, we introduce random variables (one for each agent state i of population a and location ℓ), counting how many respond. The interesting feature of this approach is that the mean field semantics is obtained by replacing the random variable with its expected value.

5.3 Example

We consider the example of Section 2.3 and model it in PEPA-S. We recall that we have m locations in the city, which we will call $\ell_1, \dots, \ell_m$. Each location has a bike station, with a certain number of bike slots, which can be either free or occupied by a bike. Each bike slot is modelled by the following two-state agent:

$$\texttt{slot\_free} \overset{def}{=} !(\text{return},\, k_{fast}).\texttt{slot\_with\_bike}; \quad \texttt{slot\_with\_bike} \overset{def}{=} !(\text{borrow}, k_{fast}).\texttt{slot\_free}$$

In the previous code, the return and borrow actions are active broadcast actions. However, they will not induce any response by passive transitions (perception function identically zero). The rate k_{fast} is a very large number, avoiding that bike slots influence the speed at which users borrow and return bikes. Users, instead, are modelled by the following three state agent:

$$\begin{gathered}
\texttt{pedestrian} \overset{def}{=} !!(\text{moveP}, k_{mp}).\texttt{soul} + !(\text{borrow}, k_B).\texttt{biker}; \\
\texttt{biker} \overset{def}{=} !!(\text{moveB}, k_{mb}).\texttt{soul} + !(\text{return}, k_R).\texttt{pedestrian}; \\
\texttt{soul} \overset{def}{=} ??(\text{arrivalP}, 1).\texttt{pedestrian} + ??(\text{arrivalB}, 1).\texttt{biker};
\end{gathered}$$

We see here how movement can be modeled in PEPA-S: processes do not actually move, but just enter a quiescent state (`soul`) in a location and use the unicast to activate a `soul` agent in the destination location. The movement rate is assumed to be constant and equal to k_{mp} for pedestrians and k_{mb} for bikers. The perception function is constructed based on the information contained in two *routing matrices* Q_p and Q_b. More specifically, $Q_p[i,j]$ is the probability that a pedestrian moves from location ℓ_i to location ℓ_j, while $Q_b[i,j]$ specifies the same probability for a biker. Hence, we have the following definition for the perception function of *moveB*:

$$u(\text{moveB}, \ell_i, \text{arrivalB}, \ell_j) = Q_b[i,j], \quad \text{and zero for any passive } \beta \neq \text{arrivalB}$$

and similarly for the pair moveP, arrivalP.

Then we construct two populations for each location: users and slots, as follows:

$$\begin{gathered}
P_{slot} ::= \mathit{slot}\{\texttt{slot\_free}[\text{s}] \parallel \texttt{slot\_with\_bike}[\text{b}]\} \\
P_{user} ::= \mathit{user}\{\texttt{pedestrian}[\text{p}] \parallel \texttt{biker}[\text{k}] \parallel \texttt{soul}[\text{a}]\}
\end{gathered}$$

Hence we have s free slots and b slots with bikes, and similarly for users. We require that $s + b = c$, the capacity of the station, and that $p+b+a = N$, the total number of users in the system. Note that most of the user population in a location will be made of souls: we need N in total in each location to properly deal with the (unlikely) situation when everybody is in a single location. Of course, these numbers may change from location to location, and specify the initial state of the model.

The population model in location ℓ_i is then

$$W_i ::= [P_{slot} \underset{\{borrow,return\}}{\bowtie} P_{user}](\ell_i),$$

giving rise to the network model

$$W_1 \| \llcorner \dots \| \llcorner W_m$$

This example can be easily extended by introducing a truck agent that moves bikes from one station to another one. In this case, however, some dependency of the perception function on the population state of the network can be useful to model redistribution policies in which bikes are taken from nearly full bike stations and brought to nearly empty ones.

Note that we did not use the broadcast in this example. This can however be helpful in other circumstances, for instance to model propagation of information through a smart bike sharing system. The actual utility of the complex broadcast communication mechanism in the context of CAS is however still unclear and deserves further investigation.

5.4 Discussion

We evaluate PEPA-S against the features discussed in Section 2.

Automated Derivation of the Model. The language supports automatic derivation of Spatial Population CTMC models and of their corresponding mean field approximations.

Agents + Environment view. Agents are modelled in a process algebra style. The notion of environment in PEPA-S is essentially embedded into the perception function. The expressivity of this choice is under evaluation.

Global and Local view. The specific introduction of a notion of agent and of a notion of population at the syntactic level of the language may facilitate the separation between the local and the global view.

Environment/Space Modeling and Awareness. The current description of the perception function assumes a static spatial structure, but it can be made in part dynamic by allowing it to depend on the population vectors of the sending and receiving locations ℓ and ℓ'. Space however is necessarily discrete. Embedding of locations in a continuous space can be indirectly done through the perception function. Space awareness can be partially embedded in the perception function, but the potential specification of adaptive behaviour for individual agents in this framework is rather limited (it may be encoded in the local state of agents, but this is highly inefficient).

Bottom-up Design. Compositionality of PEPA-S is somehow hindered by the specification of the perception function, which takes a global view and requires global knowledge. There may be compositional ways of specifying this function, but this has to be further investigated.

Modeling Adaptation and Goal-orientedness. PEPA-S has not been designed with adaptation specifically in mind, but rather as a *middle-level language*, to which we can compile CAS-SCEL models. In fact, PEPA-S features an easy derivation of mean field equations, still allowing syntactic manipulations to be performed to simplify the model. Adaptivity, though, can be partially encoded in the perception function, provided it is made more flexible (e.g. by making it dependent on the populations in each location and on the state of the passive agent).

Control over Abstraction Level. These are not features taken into account in the design of PEPA-S.

Support for Design and Analysis Techniques. Consistency of mean field approximation for very large populations should hold. There remains a scalability issue if the number of locations is large. There is no tool support for PEPA-S at the moment.

6 Stochastic HYPE

Stochastic HYPE is a process algebra developed to model three distinct types of behaviour: instantaneous, stochastic, and continuous behaviour described by ODEs over the variables of the system. Instantaneous behaviour happens when Boolean guards of events become true, and stochastic behaviour is captured in events that have exponential durations. Durations described by distributions other than exponential can be achieved by drawing from the distribution and using a timer together with an instantaneous event. Events can also reset the values of the system variables. The overall behaviour of a HYPE model can be described by multiple trajectories, one for each variable of the system. These trajectories are defined by smooth solutions to sets of ODEs interspersed with discontinuities when the system switches to a different set of ODEs and jumps where variables are reset. For additional details on HYPE, the reader is referred to [1, 11]. Stochastic HYPE is very expressive and can model space both discretely or continuously.

Prefix with influence
$$\frac{}{\langle \mathrm{a} : (\iota, r, I).E, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E, \sigma[\iota \mapsto (r, I)]\rangle}\ (\mathrm{a} \in \mathcal{E})$$

Prefix without influence
$$\frac{}{\langle \mathrm{a}.E, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E, \sigma\rangle}\ (\mathrm{a} \in \mathcal{E})$$

Choice
$$\frac{\langle E, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E', \sigma'\rangle}{\langle E + F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E', \sigma'\rangle} \qquad \frac{\langle F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle F', \sigma'\rangle}{\langle E + F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle F', \sigma'\rangle}$$

Constant
$$\frac{\langle E, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E', \sigma'\rangle}{\langle A, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E', \sigma'\rangle}(A \overset{\mathit{def}}{=} E)$$

Cooperation without synchronisation
$$\frac{\langle E, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E', \sigma'\rangle}{\langle E \underset{M}{\bowtie} F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E' \underset{M}{\bowtie} F, \sigma'\rangle}(\mathrm{a} \notin M) \qquad \frac{\langle F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle F', \sigma'\rangle}{\langle E \underset{M}{\bowtie} F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E \underset{M}{\bowtie} F', \sigma'\rangle}(\mathrm{a} \notin M)$$

Cooperation with synchronisation
$$\frac{\langle E, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E', \tau\rangle \quad \langle F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle F', \tau'\rangle}{\langle E \underset{M}{\bowtie} F, \sigma\rangle \xrightarrow{\mathrm{a}} \langle E' \underset{M}{\bowtie} F', \Gamma(\sigma, \tau, \tau')\rangle}(\mathrm{a} \in M, \Gamma \text{ defined})$$

Figure 5: Operational semantics for stochastic HYPE

6.1 Language Features

Stochastic HYPE implements separation of concerns by describing controllers/sequencers separately from subcomponents. Subcomponents capture the capabilities of the system without control and this separation allows for influences on variables to be specified in the subcomponents which are then used to derive the ODEs for the system. *Well-defined HYPE subcomponents* have the general definition

$$S(\mathcal{W}) \overset{\mathit{def}}{=} \sum\nolimits_{j=1}^{n} \mathrm{a}_j{:}(\iota, r_j, I_j(\mathcal{W})).S(\mathcal{W}) + \underline{\mathrm{init}}{:}(\iota, r, I(\mathcal{W})).S(\mathcal{W})$$

where $\mathcal{W} \subseteq \mathcal{V} = \{V_1, \ldots, V_n\}$ is the set of real variables that record the values relevant to the model. Moreoever, each a_j is distinct and different from $\underline{\mathrm{init}}$. Each a_j is an *instantaneous* ($\underline{\mathrm{a}}_j \in \mathcal{E}_d$) or *stochastic* ($\overline{\mathrm{a}}_j \in \mathcal{E}_s$) event and has an event condition $ec(\mathrm{a}) = (act(\mathrm{a}), \mathrm{restart}(\mathrm{a}))$ from a set EC. The function restart(a) is a *reset* which changes the values of $V \in \mathcal{V}$. An *activation condition* for an instantaneous event is a boolean formula over variables in $\mathcal{V}$. For a stochastic event, an activation condition is a function that describes an exponential distribution. Subcomponents contain *influences* of the form $(\iota, r, I(\mathcal{W}))$ (from the set $\mathcal{A}$) where $\iota \in \mathit{IN}$ is an *influence name*. Each well-defined subcomponent has exactly one influence name appearing across all of its influences, which only appears in that subcomponent. This name is associated with one of the model's variables via the function iv. The second and third components of an influence describe the strength of the influence and which variables will contribute to the influence in the ODE.

A *well-defined uncontrolled system* has the general definition $\Sigma = S_1(\mathcal{W}_1) \underset{*}{\bowtie} \ldots \underset{*}{\bowtie} S_s(\mathcal{W}_s)$ where each subcomponent appears exactly once. Subcomponents must synchronise on all shared events and $\underset{*}{\bowtie}$ is used to specify this. A *controller* is defined by the two-level grammar $M ::= \mathrm{a}.M \mid 0 \mid M + M$ and $\mathit{Con} ::= M \mid \mathit{Con} \underset{L}{\bowtie} \mathit{Con}$ with $\mathrm{a} \in \mathcal{E} = \mathcal{E}_d \cup \mathcal{E}_s$ and with $L \subseteq \mathcal{E}$. A controller contains no influences since it controls and sequences events only. The well-defined controlled system is $\Sigma \underset{*}{\bowtie} \underline{\mathrm{init}}.\mathit{Con}$. The first event to occur is $\underline{\mathrm{init}}$ which has *true* as its activation condition, so it triggers immediately and has the initial values as its resets. A *well-defined model* is a tuple $(\mathit{ConSys}, \mathcal{V}, \mathit{IN}, \mathit{IT}, \mathcal{E}, \mathcal{A}, ec, iv, EC, ID)$ where *ConSys* is a well-defined controlled system.

6.2 Informal Semantics

Each subcomponent in a HYPE model defines an influence for a variable and multiple subcomponents may influence a single variable. Each influence describes an additive component of the continuous change of that variable as described by the ODE given below. Events determine for each subcomponent the form of the influence. When an event happens and if a subcomponent can react to that event then

$$\begin{aligned}
ec(\underline{\text{setAngleTopRight}}_i) &= (target\_x_i \geq ux_i \wedge target\_y_i \geq uy_i, \\
&\quad angle'_i = R2D(acos(\tfrac{target\_x_i - ux_i}{dist(target\_x_i, target\_y_i, ux_i, uy_i)}))) \\
ec(\underline{\text{setAngleTopLeft}}_i) &= (target\_x_i \leq ux_i \wedge target\_y_i \geq uy_i, \\
&\quad angle'_i = 90 + R2D(acos(\tfrac{target\_y_i - uy_i}{dist(target\_x_i, target\_y_i, ux_i, uy_i)}))) \\
ec(\underline{\text{setAngleBottomLeft}}_i) &= (target\_x_i \leq ux_i \wedge target\_y_i \leq uy_i, \\
&\quad angle'_i = 180 + R2D(acos(\tfrac{ux_i - target\_x_i}{dist(target\_x_i, target\_y_i, ux_i, uy_i)}))) \\
ec(\underline{\text{setAngleBottomRight}}_i) &= (target\_x_i \geq ux_i \wedge target\_y_i \leq uy_i, \\
&\quad angle'_i = 270 + R2D(acos(\tfrac{uy_i - target\_y_i}{dist(target\_x_i, target\_y_i, ux_i, uy_i)}))) \\
ec(\underline{\text{startWalking}}_i) &= (true, speed'_i = 90) \\
ec(\underline{\text{arriveDest}}_i) &= (dist(dest\_x_i, dest\_y_i, ux_i, uy_i) \leq 100, speed'_i = 0)
\end{aligned}$$

Figure 6: Stochastic HYPE model: event conditions associated with walking

the influence that that subcomponent provides will change. Controllers specify the order in which the events happen. This can be defined formally by mapping the model to a transition-driven stochastic hybrid automaton (TDSHA) [2, 3]. These automata are a subset of piecewise deterministic Markov processes [7]. First, an operational semantics specifies the qualititative behaviour of a model as a labelled multitransition system which is then mapped to a TDSHA to express quantitative behaviour.

An *operational state* σ of the system maps each influence name ι to a pair $(r, I(\mathcal{W}))$, giving rise to a configuation $\langle ConSys, \sigma \rangle$. The operational semantics gives a labelled multitransition system over configurations (Figure 5). Two functions are used in the rules to modify the operational state. The *updating function* $\sigma[\iota \mapsto (r, I)]$ is defined in the obvious way and the *change-detecting function* Γ ensuring that there are no conflicting updates. A model has the following behaviour.

Deterministic continuous behaviour Each $\langle P, \sigma \rangle$ in the labelled multitransition system becomes a mode in the TDSHA with continuous behaviour specified by the following ODE

$$P_\sigma = \left\{ \frac{dV}{dt} = \sum \left\{ r[\![I(\mathcal{W})]\!] \;\middle|\; iv(\iota) = V \text{ and } \sigma(\iota) = (r, I(\mathcal{W})) \right\} \;\middle|\; V \in \mathcal{V} \right\}$$

Stochastic discrete behaviour A transition labelled with a stochastic event $\langle P, \sigma \rangle \xrightarrow{\overline{\text{a}}} \langle P', \sigma' \rangle$ is mapped to a stochastic transition with a *true* guard, rate $act(\overline{\text{a}})$ and reset $restart(\overline{\text{a}})$. The actual rate of transition takes into account the rates of all transitions with label $\overline{\text{a}}$.

Instantaneous discrete behaviour An instantaneous transition $\langle P, \sigma \rangle \xrightarrow{\underline{\text{a}}} \langle P', \sigma' \rangle$ is mapped to an instantaneous transition with the guard being the boolean formula $Act(\underline{\text{a}})$ and the reset being $restart(\underline{\text{a}})$.

6.3 Example

Here we present a stochastic HYPE model of a bike-sharing system in a more realistic manner using continuous space in contrast to the the one described in Section 2.3. Due to the use of continuous space, the model is somewhat larger than the preceding models which treat space discretely and therefore are more spatially abstract. We assume that the city can be represented by a $l \times w$ size map. There are m bike stations distributed in the city with (sx_i, sy_i) denoting the position of Station i (for $i = 1, ..., m$). We use ns_i and nb_i to denote the current number of available slots and bikes in Station i. Furthermore, we assume there are n bike users in the city. Users are also scattered in the city with (ux_i, uy_i) denoting the current position of the ith user (for $i = 1, ..., n$).

A bike user starts a trip and picks a random position in the city as their destination at the rate γ governed by an exponential distribution. We capture this dynamic by an event condition:

$$ec(\overline{\text{StartTrip}_i}) \;=\; (\gamma,\ dest\_x'_i = uniform(0, l), dest\_y'_i = uniform(0, w)) \quad i \in (1, \ldots, n)$$

$$
\begin{aligned}
ec(\underline{\text{pickStation\_jToStart}}_i) &= (\text{isclosest}(j, ux_i, uy_i), ssx'_i = sx_j \wedge ssy'_i = sy_j \wedge b'_{ji} = 1)\\
ec(\underline{\text{pickStation\_jToEnd}}_i) &= (\text{isclosest}(j, dest\_x_i, dest\_y_i), esx'_i = sx_j \wedge esy'_i = sy_j \wedge s'_{ji} = 1)\\
ec(\underline{\text{reserveSucceed}}_i) &= (nb_1 \geq b_{1i} \wedge ns_1 \geq s_{1i} \wedge ... \wedge nb_m \geq b_{mi} \wedge ns_m \geq s_{mi},\\
&\quad nb'_1 = nb_1 - b_{1i} \wedge ns'_1 = ns_1 - s_{1i} \wedge ... \wedge nb'_m = nb_m - b_{mi} \wedge\\
&\quad ns'_m = ns_m - s_{mi} \wedge target\_x'_i = ssx_i \wedge target\_y'_i = ssy_i)\\
ec(\underline{\text{reserveFailed}}_i) &= (nb_1 < b_{1i} \vee ns_1 < s_{1i} \vee ... \vee nb_m < b_{mi} \vee ns_m < s_{mi},\\
&\quad target\_x'_i = dest\_x_i \wedge target\_y'_i = dest\_y_i)\\
ec(\underline{\text{startCycling}}_i) &= (true, speed'_i = 200)\\
ec(\underline{\text{arriveStartStation}}_i) &= (dist(ssx_i, ssy_i, ux_i, uy_i) \leq 100, speed'_i = 0 \wedge target\_x'_i = esx_i \wedge\\
&\quad target\_y'_i = esy_i \wedge ns'_1 = ns_1 + b_{1i} \wedge ... \wedge ns'_m = ns_m + b_{mi})\\
ec(\underline{\text{arriveEndStation}}_i) &= (dist(esx_i, esy_i, ux_i, uy_i) \leq 100, speed'_i = 0 \wedge target\_x'_i = dest\_x_i \wedge\\
&\quad target\_y'_i = dest\_x_i \wedge nb'_1 = nb_1 + s_{1i} \wedge ... \wedge nb'_m = nb_m + s_{mi})\\
ec(\underline{\text{restart}}_i) &= (true, b_{1i} = 0 \wedge s_{1i} = 0 \wedge ... \wedge b_{mi} = 0 \wedge s_{mi} = 0)
\end{aligned}
$$

Figure 7: Stochastic HYPE model: event conditions associated with cycling

where $uniform(x, y)$ is a function to generate a random number between x and y. We use $(dest\_x_i, dest\_y_i)$ to denote the trip destination for User i.

Once the trip destination has been decided, the user needs to choose their travelling pattern. More specifically, we assume that if the distance between the current position and the destination is larger than a specific value d, the user will choose to travel by bike. Otherwise, the user will choose to travel by walking. This dynamic can be represented by the following event conditions:

$$
\begin{aligned}
ec(\underline{\text{chooseByWalk}}_i) &= (dist(dest\_x_i, dest\_y_i, ux_i, uy_i) \leq d,\ target\_x'_i = dest\_x_i \wedge target\_y'_i = dest\_y_i)\\
ec(\underline{\text{chooseByBike}}_i) &= (dist(dest\_x_i, dest\_y_i, ux_i, uy_i) > d,\ )
\end{aligned}
$$

where $(target\_x_i, target\_y_i)$ represents the user's current target position to move to.

If the user chooses to travel by walking, he will walk to the destination directly. This can be captured by the events conditions in Figure 6. Here the events $\underline{\text{setAngleTopRight}}_i$, $\underline{\text{setAngleTopLeft}}_i$, $\underline{\text{setAngleBottomLeft}}_i$ and $\underline{\text{setAngleBottomRight}}_i$ set the moving angle of the user directly towards the target position according to which direction the target position is relative to the user. After setting the moving angle, the user can start walking. Here, we assume that the walking speed is 90 meters per minute. Once the user arrives at their destination, they will stop there for a while and then start a new trip at the rate γ again. To allow the user to actually move, we need these subcomponents to capture the user's movement flows:

$$
\begin{aligned}
Xmove_i &\overset{def}{=} \underline{\text{init}} : (ux_i, 1, cos(D2R(angle_i)) \times speed_i).Xmove_i\\
Ymove_i &\overset{def}{=} \underline{\text{init}} : (uy_i, 1, sin(D2R(angle_i)) \times speed_i).Ymove_i
\end{aligned}
$$

where $D2R$ and $R2D$ are functions to convert degree to radius and radius to degree respectively. As can be seen from the above subcomponents, the user's position change is associated with their current speed and angle.

If the user chooses to travel by bike, we assume that the user intends to borrow a bike from the nearest bike station and return the bike to the station closest to their destination. Once the user has decided the start station from which to borrow a bike and the end station to return the bike, he will try to reserve a bike in the start station and a slot in the end station. Then, if the reservation is made successfully, the user will walk to the start station, borrow a bike there, cycle to the end station, return the bike, and walk to the destination. Otherwise, the user will directly walk to the destination if the reservation cannot be made. To capture this scenario, we first need the event conditions in Figure 7.

$$
\begin{aligned}
Con_{startTrip_i} &\overset{def}{=} \overline{\text{StartTrip}}_i.Con_{pattern_i}\\
Con_{pattern_i} &\overset{def}{=} \underline{\text{chooseByWalk}}_i.Con_{facetoDest_i} + \underline{\text{chooseByBike}}_i.Con_{pickStartStation_i}\\
Con_{facetoDest_i} &\overset{def}{=} \underline{\text{setAngleTopRight}}_i.Con_{movetoDest_i} + \underline{\text{setAngleTopLeft}}_i.Con_{movetoDest_i}\\
&\quad +\underline{\text{setAngleBottomRight}}_i.Con_{movetoDest_i}\\
&\quad +\underline{\text{setAngleBottomLeft}}_i.Con_{movetoDest_i}\\
Con_{movetoDest_i} &\overset{def}{=} \underline{\text{startWalking}}_i.Con_{\text{arriveDest}_i}\\
Con_{\text{arriveDest}_i} &\overset{def}{=} \underline{\text{arriveDest}_i}.\underline{\text{restart}_i}.Con_{startTrip_i}\\
Con_{pickStartStation_i} &\overset{def}{=} \ldots + \underline{\text{pickStation\_jToStart}}_i.Con_{pickEndStation_i} + \ldots \quad j \in (1, ..., m)\\
Con_{pickEndStation_i} &\overset{def}{=} \ldots + \underline{\text{pickStation\_jToEnd}}_i.Con_{reserve_i} + \ldots \quad j \in (1, ..., m)\\
Con_{reserve_i} &\overset{def}{=} \underline{\text{reserveFailed}_i}.Con_{facetoDest_i} + \underline{\text{reserveSucceed}_i}.Con_{facetoStartStation_i}\\
Con_{facetoStartStation_i} &\overset{def}{=} \underline{\text{setAngleTopRight}}_i.Con_{movetoStartStation_i}\\
&\quad +\underline{\text{setAngleTopLeft}}_i.Con_{movetoStartStation_i}\\
&\quad +\underline{\text{setAngleBottomRight}}_i.Con_{movetoStartStation_i}\\
&\quad +\underline{\text{setAngleBottomLeft}}_i.Con_{movetoStartStation_i}\\
Con_{movetoStartStation_i} &\overset{def}{=} \underline{\text{startWalking}}_i.Con_{arriveatStartStation_i}\\
Con_{arriveatStartStation_i} &\overset{def}{=} \underline{\text{arriveStartStation}_i}.Con_{facetoEndStation_i}\\
Con_{facetoEndStation_i} &\overset{def}{=} \underline{\text{setAngleTopRight}}_i.Con_{movetoEndStation_i}\\
&\quad +\underline{\text{setAngleTopLeft}}_i.Con_{movetoEndStation_i}\\
&\quad +\underline{\text{setAngleBottomRight}}_i.Con_{movetoEndStation_i}\\
&\quad +\underline{\text{setAngleBottomLeft}}_i.Con_{movetoEndStation_i}\\
Con_{movetoEndStation_i} &\overset{def}{=} \underline{\text{startCycling}}_i.Con_{arriveatEndStation_i}\\
Con_{arriveatEndStation_i} &\overset{def}{=} \underline{\text{arriveEndStation}_i}.Con_{facetoDest_i}\\
Con &\overset{def}{=} ... \parallel Con_{startTrip_i} \parallel ... \quad i \in (1, ..., n)
\end{aligned}
$$

Figure 8: Stochastic HYPE model: controllers

In these event conditions, isclosest(j, x, y) is a function which returns *true* if Station j is the closest station to the position (x, y). (ssx_i, ssy_i) is the position of the chosen start station whereas (esx_i, esy_i) is the chosen end station for User i. b_{ji} and s_{ji} are two tag variables which are equal to 1 if User i wants to reserve a bike or slot in Station j, otherwise they are set to 0. The event $\underline{\text{reserveSucceed}_i}$ is activated only if the available number of bikes in the start station and the available number of slots in the end station are not zero (thus greater than the tag variable). If the reservation succeeds, the target is set to the position of the start station. However, if the reservation fails, the target is set to the position of the destination. We assume that the cycling speed is 200 meters per minute. When the user arrives at the start station, the target position is set to the end station. Meanwhile, a bike will be taken from the station, thus the number of available slots in that station will increase by one. When the user arrives at the end station, the target position is set to the destination and the bike will be returned to the station, thus the number of available bikes in that station will increase by one.

To ensure all the events occur in the right sequence, we need *controllers* to impose causal or temporal constraints on the events. We first give the definition of the controller in the bike-sharing model in Figure 8. These event controllers are used to ensure the flow of events illustrated in Figure 9.

We get the uncontrolled system by synchronizing the subcomponents in the system:

$$Sys \overset{def}{=} ...Xmove_i \underset{\underline{\text{init}}}{\bowtie} Ymove_i... \quad i \in (1, ..., n)$$

Finally, the controlled system of the model is described by:

$$BikeSharingCtrl \overset{def}{=} Sys \underset{*}{\bowtie} \underline{\text{init}}.Con$$

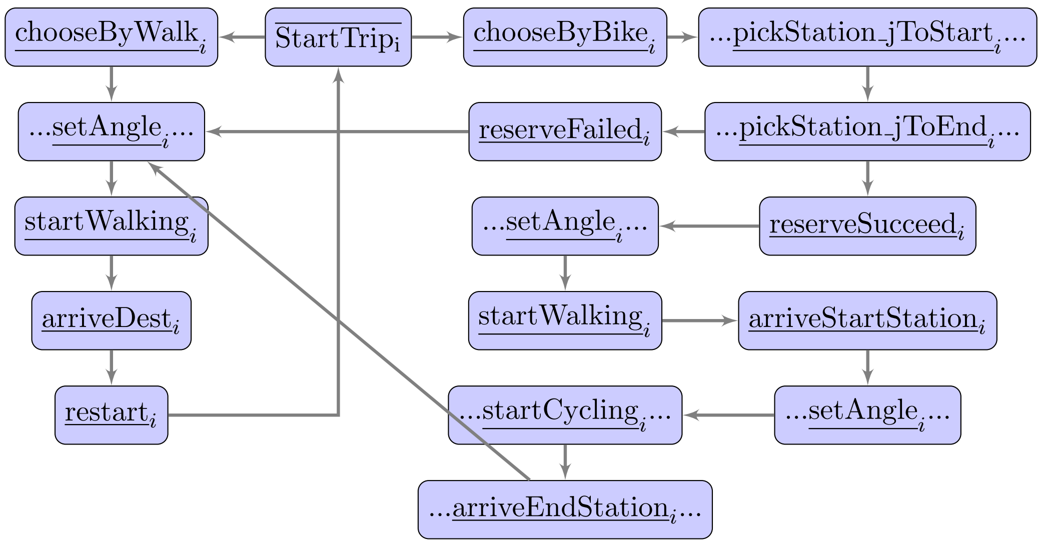
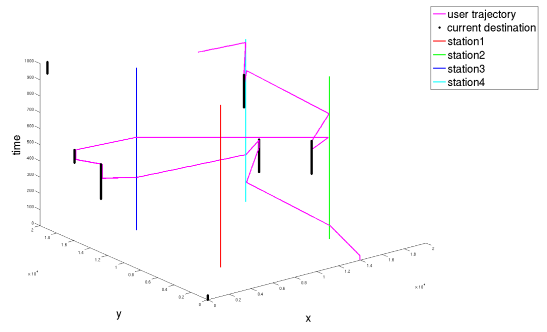
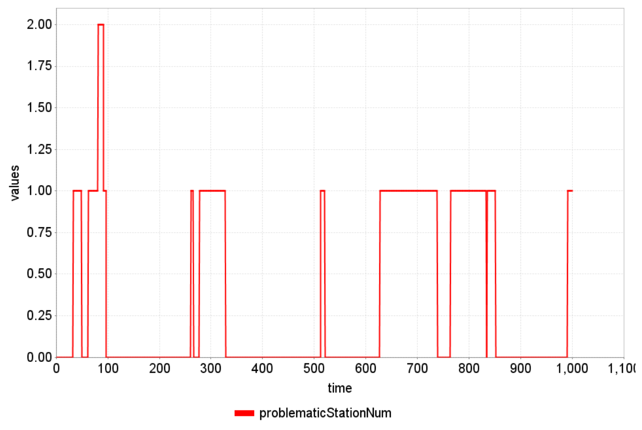


Figure 9: The flow chart of events in the bike-sharing model

In our simulation, we set the city size to 20000×20000 with four bike stations located at positions (5000,5000), (5000,15000), (15000,5000) and (15000,15000). There are 6 available bikes and 4 available slots initially in each station. Due to memory issues, our model can be loaded and simulated with at most 20 bike users. Figure 10a shows the moving trajectories of one user, and Figure 10b gives the trajectory of number of problematic stations (station with no available bikes or slots) in our simulation.



(a) The moving trajectories of one user



(b) The trajectory of number of problematic stations

Figure 10: Simulation results of the stochastic HYPE bike-sharing model

6.4 Discussion

We now consider the features of stochastic HYPE.

Automated Derivation of the Model. Stochastic HYPE has formal semantics from which a TDSHA can be obtained. The SimHyA tool can simulate stochastic HYPE models using this TDSHA.

Agents + Environment view. The environment can be modelled as a collection of subcomponents with appropriate controllers to sequence events such as the start and end of peak traffic periods. Each individual can be expressed as a collection of subcomponents and controllers to describe the behaviour of that individual as events happen. The environment is accessible by multiple agents because its characteristics are captured as global variables.

Global and Local view. Stochastic HYPE as it is defined here does not support definition of populations, however there is an extension which provides a language for generating many individuals of the same type [10]. Global variables can be used to constrain population behaviour, such as limits on growth, and to obtain global performance measures.

Environment/Space Modeling and Awareness. Continuous space can be modelled by defining functions that capture characteristics of regions in space, and variables can be associated with individuals to represent their current location in space, and describe their movement. Similarly, discrete space can be modelled by defining a function to capture the characteristics of a location, and each individual can have an associated variable for their current location. As yet, there is no technique in stochastic HYPE for translating between these different representations of space. Space-awareness, as in modelling communication within a limited range is supported by using a distance function as a guard, hence only allowing some event to occur if the distance is small enough.

To characterise spatial characteristics changing over time, subcomponents can be defined for each characteristic. However, neither stochastic HYPE nor SimHyA support partial differential equations currently, so this is more difficult for continuous space.

Bottom-up Design. Stochastic HYPE is defined in a compositional manner and thus supports this approach.

Control over Abstraction Level. As mentioned previously, the modeller has a choice in the level of abstraction for a model. However, it would be useful to develop the theory of stochastic HYPE so that a model with individual behaviour described by subcomponents can be transformed to a mean-field model with population behaviour defined by subcomponents.

Modeling Adaptation and Goal-orientedness. Stochastic HYPE is very expressive, hence adapation can be captured as functions or choice between different behaviours. It may also be possible to express goals and behaviour leading towards goals in stochastic HYPE. Alternatively, appropriate logics for hybrid systems can be considered as a way to reason about goals.

Support for Design and Analysis Techniques. Beyond the separation of capabilities and controllers, and the extension for specifying populations, there is no support for this.

7 Conclusions and Roadmap

In this deliverable we have reported the progress made on the design of the linguistic primitives of CAS-SCEL. These primitives, presented in the context of a set of design principles and interaction patters specific to CASs, have been included in four process specification languages: StocS, PALOMA, PEPA-S and Stochastic-HYPE. The explored process specification languages have been instrumental for the study of different *interaction patterns* (such as broadcast communication or anonymous interaction) that are crucial for the specification/verification and deployment of CASs. Moreover, the four considered languages considered the specification of CASs from different point of views. To show the impact of the proposed primitives on the specification of CAS, a bike sharing scenario, borrowed from one the project case studies, has been used to assess usability of the different formalisms in CAS framework.

StocS, thanks to the predicate based communication, is strongly related to the notions of *anonymity* and *dynamicity* of CASs. Indeed, the StocS predicate-based communication allows components of a system to send messages and requests to ensembles of components that are determined at execution time through the evaluation of a predicate, in a multicast fashion. Moreover, the component knowledge-base provides the realisation of various adaptation patterns, by explicit separation of adaptation data in the spirit of [5], and to model the components view on (and awareness of) the environment. Both PALOMA and PEPA-S are based on the M^2MAM framework [6], and use perception functions to represent the environment and its influence on the possible actions of agents. This mechanism is reasonably compact and, as demonstrated, amenable to scalable analysis. However, we are yet to fully explore whether its expressiveness is sufficient to capture all scenarios related to our case studies. Moreover, it is anticipated that the specification of the perception functions could become problematic when the behaviour of the environment is complex. PALOMA has been mainly focused on investigating attribute-based communication which nevertheless supports efficient and scalable analysis. As such, it has limited expressiveness and only includes indirect interaction between agents through spontaneous and induced actions. In contrast PEPA-S seeks to extend the existing stochastic process algebra PEPA, and allows different patterns of interaction and includes constructs for describing the evolution of the communication infrastructure in the modelled system. However, with respect to the QUANTICOL case studies, there is a limitation that agents are static within a single location and movement can only be mimicked by messages passed between locations instigating the "death" and "birth" of agents in respective sites. PALOMA does not suffer from this limitation.

Stochastic HYPE is expressive enough to describe continuous space, as it provides the ability to model any continuous quantity. The bike-sharing model illustrates how bike stations can be located at coordinates in the plane and how user movement can be modelled using ODEs to describe change over time. The model is larger than the other models and shows the additional complexity required to capture continuous space. For some of the QUANTICOL case studies, a more abstract view of space as discrete location may be more appropriate. The model decribes each user separately, and hence scalable analysis techniques cannot be applied directly without model transformation. Stochastic HYPE may be useful for the development of a method to transform continuous space models to discrete space models because of its expressive power, and it could also be used for models where some aspects of space is treated continuously and other aspects are treated discretely.

Relations with other WPs The work presented in this deliverable relates with the following work packages:

WP1 Emergent Behaviour and Adaptivity. This WP aims at setting up a framework to detect and control emergent behaviour of Collective Adaptive Systems (CAS). The techniques developed in WP1 will be used to support the analysis of CAS-SCEL specifications.

WP2 Collective Adaptive Behaviour in Space. The spatial representations for modelling spatial aspects of CAS considered in WP2 have influenced many of the primitives considered in this deliverable. CAS-SCEL will include specific linguistic constructs to take *spatiality* and *mobility* into account. The specification languages considered in this deliverable witness a first step towards the integration of the models of WP2 in CAS-SCEL.

WP3 Logic and Scalable Verification. The logics and techniques developed in WP3 will be used to specify and verify properties of CAS-SCEL specifications. Some of the techniques developed in WP3 should be used not only to support the verification of CAS-SCEL programs, but also at runtime to improve system adaptiveness. Indeed, an agent could use model-checking techniques, performed *on-the-fly* to support the selection of possible alternative behaviours.

WP5 Model Validation and Tool Support. To support the analysis of CAS-SCEL programs, new prototype software tools will be developed. These tools are crucial to assess the considered linguistic primitives and to use the CAS-SCEL language to specify, program and analyse significative case studies.

Work plan for the second reporting period. During the first reporting period we have mainly concentrated on CAS-SCEL language design.

In the second period we plan to continue our work within Task 4.1: CAS-SCEL language design and to use the syntactic constructs considered in this deliverable to distill a single specification/programming language. In the definition of our language we will also consider other approaches, like for instance the one based on *nature-inspired* coordination mechanisms [23] [19], that guarantees abstraction from identity, scalability, ability to plug-in/out new components without affecting the behaviour of the system.

We will also start working on the implementation CAS-SCEL within Task 4.2: Programming collective adaptive systems. The starting point of this work will be jRESP, the Java runtime environment that can be used to execute and simulate SCEL programs. We will consider a porting of the code to the Eclipse Modelling Framework (EMF).

The tool developed in Task 4.2 will be one of the first step towards the integration of CAS-SCEL with the analysis techniques developed within WP1 and WP3. This integration will be pursued within Task 4.3: Design workflow and analysis pathway and will lead to an integrated framework that will enable usage of CAS-SCEL and of the related logics and tools by researchers who are not experts in formal methods, verification procedures, and quantitative analysis techniques.

References

[1] L. Bortolussi, V. Galpin, and J. Hillston. HYPE with stochastic events. In *Proceedings of the Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL 2011)*, EPTCS 57, pages 120–133, Saarbrücken, 2011.

[2] L. Bortolussi and A. Policriti. Hybrid dynamics of stochastic programs. *Theoretical Computer Science*, 411:2052–2077, 2010.

[3] L. Bortolussi and A. Policriti. (Hybrid) automata and (stochastic) programs: The hybrid automata lattice of a stochastic program. *Journal of Logic and Computation*, 23:761–798, 2013.

[4] Dario Bruneo, Marco Scarpa, Andrea Bobbio, Davide Cerotti, and Marco Gribaudo. Markovian agent modeling swarm intelligence algorithms in wireless sensor networks. *Performance Evaluation*, 69(3-4):135–149, March 2012. 00011.

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

[6] Davide Cerotti, Marco Gribaudo, Andrea Bobbio, Carlos Miguel Tavares Calafate, and Pietro Manzoni. A markovian agent model for fire propagation in outdoor environments. In Alessandro Aldini, Marco Bernardo, Luciano Bononi, and Vittorio Cortellessa, editors, *EPEW*, volume 6342 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.

[7] M.H.A. Davis. *Markov Models and Optimization.* Chapman & Hall, 1993.

[8] R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A formal approach to autonomic systems programming: the SCEL Language. *ACM Transactions on Autonomous and Adaptive Systems*, 2014. To appear. Available as Technical Report from `http://eprints.imtlucca.it/2117/`.

[9] Rocco De Nicola, Gian Luigi Ferrari, Michele Loreti, and Rosario Pugliese. A language-based approach to autonomic computing. In Bernhard Beckert, Ferruccio Damiani, Frank S. de Boer, and Marcello M. Bonsangue, editors, *FMCO*, volume 7542 of *Lecture Notes in Computer Science*, pages 25–48. Springer, 2011.

[10] C. Feng. *Modelling opportunistic networks with HYPE.* MSc dissertation, School of Informatics, University of Edinburgh, 2012.

[11] V. Galpin, L. Bortolussi, and J. Hillston. Hype: Hybrid modelling by composition of flows. *Formal Aspects of Computing*, 25(4):503–541, 2013.

[12] Vashti Galpin, Jane Hillston, and Federica Ciocchetta. A semi-quantitative equivalence for abstracting from fast reactions. In Ion Petre and Erik P. de Vink, editors, *CompMod*, volume 67 of *EPTCS*, pages 34–49, 2011.

[13] 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. bibtex: PA:BradleyHaiden:2010:FluidFrameworkPEPA.

[14] Holger Hermanns and Joost-Pieter Katoen. The how and why of interactive markov chains. In Frank S. de Boer, Marcello M. Bonsangue, Stefan Hallerstede, and Michael Leuschel, editors, *FMCO*, volume 6286 of *Lecture Notes in Computer Science*, pages 311–337. Springer, 2009.

[15] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996. bibtex: PA:Hillston:1996:CompositionalPerformanceModeling.

[16] Jane Hillston. Fluid flow approximation of PEPA models. In *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, pages 33–42, 2005. 00226.

[17] Diego Latella, Michele Loreti, Mieke Massink, and Valerio Senni. Stochastically timed predicate-based communication primitives for autonomic computing. In N. Bertrand and L. Bortolussi, editors, *Proceedings of 12th Quantitative Aspects of Programming Languages and Systems (QAPL)*, Electronic Proceedings in Theoretical Computer Science, page To appear., 2014.

[18] Diego Latella, Michele Loreti, Mieke Massink, and Valerio Senni. Stochastically timed predicate-based communication primitives for autonomic computing - full paper. Technical Report TR-QC-02-2014, QUANTICOL Project, 2014. `http://www.quanticol.eu/`.

[19] Andrea Omicini and Mirko Viroli. Coordination models and languages: from parallel computing to self-organisation. *Knowledge Eng. Review*, 26(1):53–59, 2011.

[20] K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2-3):285–327, 1995.

[21] A. Romanel, L. Dematte, and C. Priami. The beta workbench. Technical Report TR-03-2007, Center for Computational and Systems Biology, Trento, 2007. bibtex: SB:Priami:2007:BetaWorkbench.

[22] Mirco Tribastone, Stephen Gilmore, and Jane Hillston. Scalable differential analysis of process algebra models. *Software Engineering, IEEE Transactions on*, 38(1):205–219, 2012. 00041.

[23] Mirko Viroli, Matteo Casadei, Sara Montagna, and Franco Zambonelli. Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*, 6(2):14:1–14:24, June 2011.