ascens

www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D2.1: First Report on WP2

Enhanced Connectors, Resource-Aware Operational Models and the Negotiate-Commit-Execute Schema and its Foundations

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **7.6.2010**

Lead contractor for deliverable: **UNIPI**
Author(s): **S. Bensalem (UJF-Verimag), M. Boreale & M. Loreti (UDF), R. Bruni & A. Corradini & F. Gadducci (ed.) & U. Montanari & M. Sammartino (UNIPI), M. G. Buscemi & R. De Nicola & A. Lluch Lafuente & A. Vandin (IMT), G. Cabri (UNIMORE), D. Latella & M. Massink (ISTI), M. Hölzl (LMU)**

Due date of deliverable: **September 30, 2011**
Actual submission date: **November 15, 2011**
Revision: **Final**
Classification: **PU**

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

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

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In this deliverable, we report on the WP2 activities conducted in months 1–12.

In Task 2.1 our results advanced the theoretical foundations of connectors and their distributed implementation, and the use of process calculi for resource-aware networks and stochastic systems.

In Task 2.2 we tackled the issue of formal models of autonomic systems by proposing a calculus for contract distillation by distributed synthesis designed along the Negotiate-Commit-Execute (NCE) scheme of interaction, a general system model for ensembles (GEM) and a definition of "black-box" adaptation based on this model, and a conceptual framework for "white-box" adaptation, conceived around a prominent role of clearly and neatly identified *control data*: computational data that govern the execution and are conveniently managed to enact adaptive behaviors.

In Task 2.3 we formalized and investigated emerging behaviors of autonomic systems by applying game paradigms in service composition and developing coalgebraic techniques for dynamic systems.

The content of the deliverable is organized around the above structure of themes.

Contents

1 Introduction

During months 1–12, we have been working on all of the three tasks connected to the work package 2. We summarize below our contributions to the package: a more detailed description for the contributions to each tasks, as well as a larger reporting of each line of research, can be found in subsequent sections. For each of the tasks the activities can be roughly divided along two main strands of research.

Task 2.1 (Resource-aware operational models) The overall aim of the task is concerned with "the study of resource-aware infrastructures and networking middleware modeled in terms of advanced components, glues and connectors which can support different levels of guarantees, reliability, dynamicity and integration to heterogeneous components."

Along those lines, for the advances occurred in the first year we identified two main lines of research. Some of the more mature work concerned the strand on *Foundations of resource-aware connectors*. One of the aims of the task was to overcome the fragmentation in the body of knowledge and the different notions and terminologies involving connectors. We thus proceeded to overview and compare some of the most notable theories, defining mutual embeddings and planning enhancements. We then tackled the specific case of our main component framework, based on BIP. As promised, a theory for its distributed implementation has been developed: its main results are reported here.

The activities in the second strand has concerned *Advanced models of networking middleware*, the emphasis being put on the development of novel frameworks based on process calculi. So, we proposed a network-aware extension of classical π-calculus (NCPi), whose syntax allows expressing the creation and the activation of connections, and whose semantics shows multisets of routing paths that are covered concurrently. Also, we addressed the development of a uniform framework for stochastic calculi, in order to provide a general, SOS-like language definitions for the class of transition system with multiplicity, in relation with the rate condition principle of Continuous Time Markov Chains.

Task 2.2 (Adaptive SCs: building emergent behavior from local/global knowledge) The research on this task should be concerned with the "develop[ment of] robust mathematical foundations for interaction scenarios [...] address[ing] models that can favor a mixture of static and dynamic analysis tools by adhering to the NCE schema".

At the end of the first year, also the work on this task has two different facets. Concerning the development of models for interaction scenario that adhere to the Negotiate-Commit-Execute (NCE) schema, we focused on *Contract distillation by distributed synthesis*, by proposing a calculus where the distillation of the contract in a client-server scenario can be obtained by composing the distributed information provided directly by each participant. The addressing of such a distillation was explicitly mentioned in the Description of Work. On a more foundational issue, we planned to tackle the identification of *Conceptual models for autonomicity*. More specifically, in order to characterize the core nucleus identifying emergent behavior, we presented two frameworks for adaptivity. The first one is concerned with "black-box" adaptation and based on the denotational GEM system model for ensembles. Black-box adaptation describes how well a system can perform in various environments without looking at the internal mechanisms used to achieve adaptation. The second framework is focused on the mechanics of adaptation and conceived around a prominent role of *control data*: computational data that are managed to enact adaptive behaviors. Most programming paradigms proposed for adaptive computation are shown to fit within these two frameworks.

Task 2.3 (Modeling SCEs with collaborative and competitive behavior) The research on this last task has the ambition of "develop[ing] a theory combining as much as possible the flexibility and compositionality of computer science semantic models with the expressiveness of models developed

by economic theorists", possibly with an in-depth analysis of the "adapt[ion] and re-use in this context many [of] co-algebraic tools well-known for ordinary transition systems".

Also this task has two different prongs. On the one hand we report on the application of *Game paradigms for service composition*. At first, a repeated non-cooperative job scheduling game has been investigated, whose players are Grid sites and whose strategies are scheduling algorithms, showing whether different strategies may reach a Nash equilibrium or not. Also, the application of adaptation in service composition was investigated, as represented by a peer-to-peer energy management scenario. The latter is modeled by exploiting an instance of the minority game (a family of games where the choice of the minority wins): the system was implemented by Jade, and its outcome assessed.

On the other hand, we describe more foundational contributions focusing on *Coalgebraic techniques for dynamical systems*. Referring to analysis tools mentioned in the Description of Work, it pushed the coalgebraic view of weighted automata: a generalization of non-deterministic automata where each transition has also a quantity expressing the cost of its execution, drawn from a semiring.

2 On Task 2.1: Resource-aware operational models

As we already mentioned, for the advances occurred in the first year in Task 2.1we focused on two main lines of research. The first concerns the *Foundations of resource-aware connectors*, addressing the fragmentation in the body of knowledge and the many different notions and terminologies involving connectors, as well as the development of a suitable paradigm for distributed connectors. The other focused on *Advanced models of networking middleware*, the emphasis being put on the development of novel frameworks based on process calculi.

2.1 Task 2.1: strand on Foundations of resource-aware connectors

Software architectures are essential for mastering the complexity of systems and easing their analysis. They allow a separation between the detailed behavior of components and their overall coordination. Coordination is often expressed by constraints that define possible interactions between components.

The term *connector* denote entities that can regulate the interaction of a collection of components [PW92]. In fact, component-based design relies on the separation of concerns between coordination and computation: the *components* are loosely coupled sequential computational entities that come equipped with a suitable interface (e.g. comprising the number, kind and peculiarities of communication ports) and the *connectors* can be regarded as (suitably decorated) links between the ports of the components. Semantically, each connector can impose suitable constraints on the communications between the components it links together (e.g. handshaking, broadcasting, multicasting). Then, the evolution of an ensemble can be seen as if played in rounds: at each round, the components try to interact through their ports and the connectors allow/disallow some of the interactions selectively.

Foundations of connectors Recent years witnessed the development of different mathematical frameworks that are used to specify, design, analyze, compare, prototype and implement suitable connectors. A rigorous mathematical foundations is crucial for the analysis of coordinated distributed systems, and recent years have witnessed an increasing interest about a rigorous modeling of (different classes of) connectors. Due to the high dynamicity of autonomic component ensembles, more powerful classes of connectors are needed than those available in the literature, to be empowered with mechanisms for resource- and network-awareness (the behavior of a connector may depend on the links it is tied to, e.g. for optimizing the routing of messages), as well as adaptation, reflection and reconfigurability.

One of the main limitations of the state-of-the-art theories of connectors is the lack of a reference paradigm for describing and analyzing the information flow to be imposed over components

for proper coordination. Such a paradigm would allow designers, analysts and programmers to rely on well-founded and standard concepts instead of using all kinds of heterogeneous mechanisms, like semaphores, monitors, message passing primitives, event notification, remote call, etc. Moreover, the reference paradigm would facilitate the comparison and evaluation of otherwise unrelated architectural approaches as well as the development of code libraries for distributed connectors. To some extent, the reference paradigm could thus play the role of a unifying semantic framework for connectors.

Concerning the fragmentation issue, during the first year we overviewed and compared some notable theories of connectors, defined some mutual embeddings and planned some possible enhancements. Our results suggest that the *tile model* can provide a suitable semantic framework for connectors, thanks to its generality and flexibility. In fact it helped relating several formal frameworks of connectors that are otherwise very different in style and nature (CommUnity, Reo, Petri nets with boundaries, wire calculus). First, the tile model can naturally account for several forms of dynamicity in the otherwise static structure of connectors. Moreover, we believe that when the algebraic properties offered by the tile model are missing, then it becomes cumbersome to account for concurrency aspects in a standard, reliable manner. Finally, while compositionality often needs ad hoc proofs when other approaches are considered, in the case of the tile model, it can be expressed in terms of the congruence result for tile bisimilarity and it can be guaranteed by the format in which basic tiles are presented.

Foundations of dynamic architectures There exists a large number of formalisms supporting a concept of architecture, including software component frameworks, systems description languages and hardware description languages. Despite an abundant literature and a considerable volume of research, there is no agreement on a common concept of architecture, while most definitions agree on the core e.g. diagrammatic representations by using connectors. This is due to two main reasons.

The first is the lack of rigorous operational semantics defining architectures via composition operators, i.e., the behavior of a composite component is inferred from the behavior of its constituents by applying architectural constraints. For existing component frameworks, the definition of rigorous operational semantics runs into many technical difficulties. They fail to clearly separate between behavior of components and architecture. Connectors are not just memoryless switching elements. They can be considered as special types of components with memory e.g. fifo queues and specific behavior. Another difficulty stems from verbose architecture definitions e.g. by using ADLs [MT97], that do not rely on a minimal set of concepts. Such definitions are hardly amenable to formalization. Finally, some frameworks [GMW97] use declarative languages to express global architecture constraints which are useful for checking correctness but cannot provide a basis for defining operational semantics.

The second is the distinction between static and dynamic architectures. Usually, hardware and system description languages rely on static architecture models. The relationships between components are known at design time and are explicitly specified as a set of connectors defining possible interactions. Dynamic architectures are needed for modeling reconfigurable systems or systems that adapt their behavior to changing environments. They are defined as the composition of dynamically changing architecture constraints offered by their constituent components. Filling the gap between static and dynamic architecture models raises a set of interesting problems. In principle, dynamic architecture models are more general: each configuration corresponds to a static architecture model. Is it possible to define a dynamic architecture modeling language as an extension of a static architecture modeling language? Furthermore, if we restrict to systems with a finite - although potentially large - set of possible configurations, any dynamic architecture model can be translated into a static architecture model. Such a translation can yield very complex static architecture models. As a rule, using dynamic architectures may lead to more concise models. However, static architecture models can be executed more efficiently thanks to the global knowledge of connectors [ADG98].

2.1.1 Connectors for P/T nets and BIP (Task 2.1)

During the first year of the project the research on the foundations of connectors looked for a suitable general framework for their description, moving from Petri nets towards tiles.

Petri nets for BI(P) We investigated the expressive power of BIP w.r.t. different classes of connectors in the literature. The main contribution shows that BI(P), the BIP component framework without priorities, is as expressive as the *Petri nets with boundaries*, introduced by Pawel Sobocinski in [Sob10]. As a byproduct of [Sob09], BI(P) is also equivalent to the so-called *Petri calculus*, a dialect of the *wire-calculus* [Sob09], which thus provides a basic process algebra for BI(P) systems.

Technically speaking, the contribution in [Sob10] can be summarized as follows. Nets with boundaries are first introduced, taking inspiration from the open nets of [BCEH05]. Nets with boundaries can be composed in series and in parallel and come equipped with a labeled transition system that fixes their operational and bisimilarity semantics. Then, a suitable instance of the *wire calculus* from [Sob09] is presented, called *Petri calculus*, that roughly models circuit diagrams with one-place buffers and interfaces. The first result enlightens a tight semantics correspondence: it is shown that a Petri calculus process can be defined for each net such that the translation preserves and reflects operational semantics (and thus also bisimilarity). The second result provides the converse translation, from Petri calculus to nets, which requires some technical ingenuity.

Another contribution has been the generalization of the work in [Sob10] carried out in [BMM11a], where it is shown that if the tile model is used in place of the wire calculus then the translation from Petri calculus to nets becomes simpler and that the main results of [Sob10] can be extended to P/T Petri nets with boundaries, where arcs are weighted and places can contain more than one token. The results in [Sob10, BMM11a] are particularly significant because they provide, for the first time, a minimal algebra of (P/T) Petri nets, a task that has been attempted several times in the past, but typically lead to cumbersome models or to frameworks allowing to build terms with no (P/T) Petri net counterpart.

Finally, we have exploited the tile model as a unifying framework to compare BI(P) with other models of connectors and to propose suitable enhancements of BI(P).

The research thread illustrated above stem from previous work on stateless connectors. We recall that a connector is called *stateless* when the interaction constraints it imposes over its ports stays the same at each round; it is called *stateful* otherwise. In [BLM06], an algebra of stateless connectors was presented that was inspired by previous work on simpler algebraic structures [BGM02, Ste98]. It consists of five kinds of basic connectors (plus their duals), namely symmetry, synchronization, mutual exclusion, hiding and inaction. The connectors can be composed in series or in parallel. The operational, observational and denotational semantics of connectors are first formalized separately and then shown to coincide. Moreover, a complete normal-form axiomatization is available for them. These networks are quite expressive: for instance it is shown [BLM06] that they can model all the (stateless) connectors of the architectural design language CommUnity [FM97].

In [Sob10, BMM11a] the algebra of stateless connectors is extended with two constants, representing the empty/filled one-position buffer, with the key difference that the semantics for the buffer in [BMM11a] accommodates for the asynchronous reception (and storage) of an unbounded number of data. A similar extension was available in [ABC$^+$09], where it was used to relate *Reo* with tiles.

Reo [Arb04] is an exogenous coordination model for software components. Reo is based on channel-like connectors that mediate the flow of data and signals among components. Notably, a small set of point-to-point primitive connectors is sufficient to express a large variety of interesting constraints over the behavior of connected components, including various forms of mutual exclusion, synchronization, alternation, and context-dependency. Typical primitive connectors are the synchronous / asynchronous / lossy channels and the asynchronous one-place buffer. They are attached

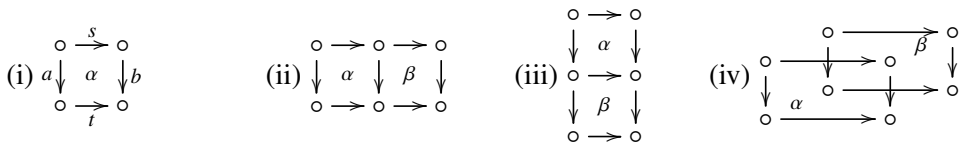


Figure 1: Examples of tiles and their composition

to ports called Reo nodes. Components and primitive connectors can be composed into larger Reo circuits by disjoint union up-to the merging of shared Reo nodes.

Summing up the results in [Sob10, BMM11a, BMM11b, ABC+09] we obtain an equivalence between BI(P) and the subclass of Reo circuits with (one-bounded) one-position buffers, via mutual encoding in the *tile model*. Notably, the basic set of connectors we need are the stateless ones of [BLM06] plus one-position buffers.

Tiles for connectors The Tile Model [GM00, Bru99] offers a flexible semantic setting for concurrent systems [MR99, FM00, BM02] and for defining the operational and abstract semantics of connectors.

The name 'tile' is due to the graphical representation of such rules (see Fig. 1). A tile $\alpha : s \xrightarrow[b]{a} t$ is a rewrite rule stating that the *initial configuration* s can evolve to the *final configuration* t via α, producing the *effect* b; but the step is allowed only if the 'arguments' of s can contribute by producing a, which acts as the *trigger* of α (see Fig. 1(i)). Triggers and effects are called *observations* and tile vertices are called *interfaces*. Tiles express the reactive behavior of connectors in terms of $\langle$trigger, effect$\rangle$ pairs of labels. In this context, the usual notion of bisimilarity over the derived Labeled Transition System is called *tile bisimilarity*. The algebra of stateless connectors in [BLM06] can be regarded as a kind of tile model where all basic tiles have identical initial and final connectors.

Tiles compose horizontally, in parallel, or vertically to generate larger steps. Horizontal composition $\alpha;\beta$ coordinates the evolution of the initial configuration of α with that of β, yielding the 'synchronization' of the two rewrites (see Fig. 1(ii)). Horizontal composition is possible only if the effect of α provides the trigger for β. Vertical composition is the sequential composition of computations (see Fig. 1(iii)). Parallel composition builds concurrent steps (see Fig. 1(iv)). Tile bisimilarity is a congruence (w.r.t. series and parallel composition) when a simple format is met by basic tiles [GM00].

Roughly, the semantics of component-based systems can be expressed via tiles when: i) components and connectors are equipped with sequential composition $s;t$ (defined when the output interface of s matches the input interface of t), with identities for each interface and with a monoidal tensor product $s \otimes t$ (associative, with unit and distributing over sequential composition); ii) observations have analogous structure $a;b$ and $a \otimes b$. Technically, we require that configurations and observations form two monoidal categories with the same underlying monoid of objects.

Related works The problem of interpreting BIP interaction models (i.e., the second layer of BIP) in terms of connectors has been addressed for the first time in [BS08a], where BIP interaction is described as a structured combination of two basic synchronization primitives between ports: rendezvous and broadcast. In this approach, connectors are described as sets of possible interactions among involved ports. In particular, broadcasts are described by the set of all possible interactions among participating ports and thus the distinction between rendezvous and broadcast becomes blurred. The main drawback of this approach is that it induces an equivalence that is not a congruence. The paper [BS10] defines a causal semantics that does not reduce broadcast into a set of rendezvous and tracks the causal dependency relation between ports. This is shown to correspond to the Algebra of Causal Interaction Trees, that comprises a causality operator and a parallel composition operator. In the initial model, terms are

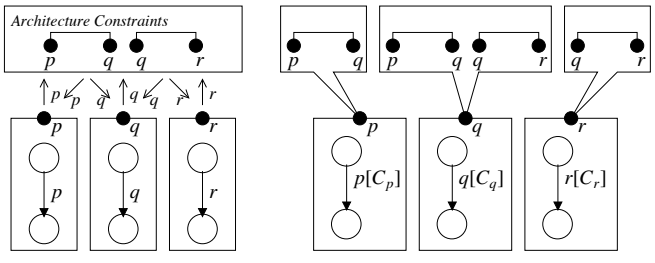
sets of trees, where the successor relation represents causal dependency between interactions. Notably, the causal semantic equivalence is a congruence w.r.t. such two operations.

Conclusions and on-going work The comparison between the Petri calculus and the tile model led us to infer that the tile vertical composition is better suited than the one in the Petri calculus (and more generally in the wire calculus) when concurrent systems are considered. Technically, the difference relies in the vertical tile composition being monoidal, while this is not so for the wire calculus.

Some interesting research avenues for future work are (i) the comparison between Reo and BIP connectors, in particular, the study of suitable extensions of BIP interaction model accounting for stateful connectors; (ii) the representation of priorities in approaches such as the algebra of connectors and the tile model; and (iii) the identification of suitable classes of connectors with link creation and consumption along the lines of [BM02].

2.1.2 The dynamic component-based framework Dy-BIP (Task 2.1)

In the past year we proposed the Dy-BIP component framework based on rigorous operational semantics for modeling both static and dynamic architectures. Dy-BIP can be considered as an extension of the BIP language for the construction of composite hierarchically structured components from atomic components. These are characterized by their behavior specified as automata extended with data and functions described in C. A transition of an automaton is labeled by a port name, a guard (boolean condition on local data) and an action (computation on local data). In BIP, architectures are composition operators on components defining their interactions. An interaction is described as a set of ports from different components. It can be executed if there exists a set of enabled transitions labeled by its ports. The completion of an interaction is followed by the completion of the involved transitions: execution of the corresponding actions followed by a move to the target state. An operational semantics for BIP has been defined in [BBBS08]. It provides a basis for the implementation of an execution Engine that orchestrates component execution. The Engine knows the set of the interactions modeling the architecture. It executes cyclically and atomically the following three-step protocol: 1) from a state each component sends to the Engine the ports of its enabled transitions; 2) the Engine computes the set of feasible interactions (sets of received ports corresponding to some interaction); 3) the Engine chooses non-deterministically one interaction amongst the feasible interactions by sending back to the components the names of their ports involved in this interaction. Figure 2(a) illustrates a statically defined architecture defined by interactions pq and qr. Its consists of three components offering communications through ports p,q and r.



(a) Static architecture (b) Dynamic architecture

Figure 2: Static and dynamic architecture.

In contrast to BIP, the set of interactions characterizing architectures in Dy-BIP changes dynamically with states. A port p has an associated architecture constraint C_p which describes possible sets of

interactions involving p. Feasible interactions from a state are computed as solutions of the constraint obtained as the conjunction of the constraints of all the enabled transitions. Figure 2(b) illustrates a dynamic architecture with three components offering ports p, q and r with associated constraints C_p, C_q and C_r. As for the static architecture of this figure, the possible interactions are pq and qr.

We provided a formalization of the operational semantics for Dy-BIP. Its implementation consists of an execution Engine which as for BIP, orchestrates components by executing atomically a three-step protocol. The protocol differs in that components send not only port names of enabled transitions but also their associated architecture constraints.

Dy-BIP is an extension of BIP. A BIP model with a global architecture constraint C, can be represented as a Dy-BIP model such that the constraint C_p associated with a port p is the set of the interactions of C involving p.

Dy-BIP allows modeling dynamic architectures as the composition of instances of component types. For the sake of simplicity, we assume that there is no dynamic creation/deletion of component instances. The main results are:

- Definition of a language for the description of architecture constraints. The language is expressive and amenable to analysis and execution. It defines formulas of a first order logic allowing quantification over component types. Formulas characterize sets of interactions. They involve port names used as logical variables. Given a formula, a feasible interaction is any set of ports assigned true by a valuation which satisfies the formula.

- Study of a semantic model and an associated modeling methodology for writing architecture constraints associated with ports. For a port p, the associated constraint is decomposed into a causal constraint and a filter constraint. A causal constraint expresses three types of dependency characterizing interaction between ports [BS08b]: “absence of dependency”, “must interact”, “may interact”. In contrast to causal constraints which enforce interactions, filter constraints are invariants used to discriminate undesirable configurations of a component’s environment.

- Implementation principles for Engines handling symbolic architecture constraints. The proposed implementation is based on the resolution architecture constraints on the fly. The Engines use efficient constraint resolution techniques based on BDDs. For a given model, quantifiers over components can be eliminated and formulas become Boolean expressions on ports. Notice that, another possible implementation could be done based on the resolutions of global architecture constraints after translation into a static architecture model.

- Experimental results and benchmarks showing benefits of using dynamic architectures compared to static architectures. We consider several examples showing that compositional modeling of dynamic architectures allows enhanced conciseness and rigorousness. In particular, it is possible to master complexity of intricate dynamic interaction by focusing on the capabilities for interaction of individual components.

In contrast to other frameworks [IW95, Mét98], Dy-BIP relies on a clear distinction between behavior and architecture as a set of stateless architecture constraints characterizing interactions. These constraints are specified compositionally as the conjunction of individual architecture constraints of components. Frameworks usually describe dynamic architectures as a set of global transitions between configurations. Only process algebras adopt a compositional approach: Nonetheless, they do not encompass a concept of architecture as behavior and composition operators are intermingled.

Dy-BIP differs from other formalisms such as [MK96] in that it has rigorous operational semantics.

In [GMW97], a First order logic extended with architecture-specific predicates is used. However, there is no separation between two basic synchronization protocols (rendezvous and broadcast)

whose combination is expressive enough to represent any kind of interaction and avoids the exhaustive enumeration of all possible rendezvous [BS08a].

In [KG10], a dynamic architecture is defined as a set of global transitions between global configurations, whereas in Dy-BIP the global configuration is computed at runtime from the local constraints of each component. Constraints in [KG10] are expressed in a First Order Logic extended with architecture-specific predicates, whereas in Dy-BIP, constraints are stateless (they are based on the boolean representation of causal rules [BS08b]) and take advantage of the stateful behavior of the components by eliminating some of the undesirable global configurations implicitly.

In [BD07] an operational semantics is provided as well as the composition of global configurations from local ones. It uses rewriting logic for representing dynamic systems using three forms of dependencies between services (mandatory, optional and negative). Nonetheless, as dynamism is supported only at the installation phase, they assume that each component provides many services all the time without any notion of state on each component's behavior.

Other formalisms provide dynamism only at the deployment phase as they focus on enabling the modification of components without taking off-line the whole system [BCDW04]. Although Dy-BIP does not support the creation/deletion of components, this functionality can be achieved by using a pool of components and writing the constraints that provide at runtime the interactions with components of the right component type.

2.2 Task 2.1: strand on Advanced models of networking middleware

Traditional formalisms for the description of distributed systems, such as the π-calculus, abstract away details of the communication infrastructure. This is not adequate for systems where behavior depends on various kinds of resources, such as swarm intelligent systems, where agents must take into account the available resources (e.g. amount of battery charge, signal strength...) when planning their actions, or cloud computing systems, made of infrastructural elements with different policies, bandwidth and access rights. Robot ensembles are examples of the former kinds of systems: robots communicate through the ad-hoc wireless network they form; this network may change whenever the quantity of resources available to robots changes, e.g. a robot may stop communicating due to low battery charge.

The need to capture a wide range of resources in the same theoretical framework led us to consider presheaf models, categorical models where the structure of resources is decoupled from the way resources are used, namely: resources and their operations form a category C, and the abstract syntax and semantics are given in terms of functors from C to the category Set that associate resources with the processes using them and with the behavior enabled by that quantity of resources.

The efficient treatment of resources is a key issue in verification, because the number of states can become infinite when fresh resources are produced. Presheaf models are not finitistic, but coalgebras over a broad class of presheaves can be implemented as HD-automata [CM10, CKM10], more concrete operational models that allows for name deallocation and hence are suitable for verification.

In months 1–12, first we considered presheaf models, that are algebraic/coalgebraic categorical models where the structure of resources is decoupled from the way resources are used. Then we defined a network-aware extension of the π-calculus (NCPi) [MS11], whose syntax allows expressing the creation and the activation of connections, and whose semantics shows multisets of routing paths that are covered concurrently. Finally, we investigated how NCPi relates to SCEL.

To integrate qualitative descriptions with quantitative ones in a uniform way within a single mathematical framework, in months 1–12 we have introduced a uniform framework for the definition of Stochastic Process Calculi (SPCs). The framework is based on the notion of *State to Function Labeled Transition Systems* (FuTSs) and allows for an SOS-like language definition, providing a simple and

elegant solution to several issues typical of this class of languages. The approach has been applied to the definition of significant fragments of major SPCs proposed in the literature.

The main aim of SPCs is the integration of qualitative descriptions with quantitative (especially performance) ones in a single mathematical framework by building on the combination of Labeled Transition Systems (LTSs) and Markov models—usually CTMCs. In a typical SPC, prefix-operators are enriched with rates of exponentially distributed random variables (RVs) characterizing their duration. Although the same class of RVs is assumed in most SPCs, models and techniques underlying such calculi turn out to be significantly different in many respects. Some differences are *conceptual*, e.g. the *process interaction paradigm*, the *association* of rates with actions or not, the *rate of synchronizations*. Other differences, instead, are purely *technical*. The prominent example of such a situation is the modeling of the CTMCs *race condition* principle and its relationship to *transition multiplicity*.

2.2.1 A framework for resource-aware process calculi (Task 2.1)

As a concrete case-study for our investigation, we defined a network conscious, proper extension of the π-calculus, called NCPi [MS11]. In particular:

- We distinguish two types of names: *sites*, which are the nodes of the network, and *links*, named connectors between pairs of sites. Sites are just atoms, e.g. a, links have the form l_{ab}, meaning that there is a link named l between a and b. The syntax has new primitives for handling links and, since it is no more required for processes to communicate on shared channels, an extended output primitive is introduced that specifies not only the emission site but also the destination one.

- We provide two semantics: an interleaving one, inspired by the π-calculus early semantics, where an observation is a sequence of links representing the observable part of a routing path, and a concurrent one, where concurrent transmissions can be observed in the form of a multiset of paths. Restricted connector names do not appear in the observations, which thus can possibly be as abstract as in the π-calculus.

- The behavioral equivalence in the interleaving case is not preserved by the input prefix, as in the π-calculus, but in the concurrent case it is preserved by all operators, hence it is a *congruence*. This is because the concurrent semantics provides more observations than the interleaving one, resulting in a finer, compositional bisimilarity.

Now we give an overview of NCPi through a motivating example. We consider the system made of a network manager M, using a reserved site m, and two processes p and q, which access the network respectively through the sites a and b. The manager is the only entity that can create new links and grant access to them. The process p wants to send a message to q, but we assume that there are no links between a and b allowing p and q to communicate. The processes are

$$\begin{array}{lll} M & = m(x).m(y).(l_{xy})(\overline{m}xl_{xy}.M) & \qquad q = b(x).q' \\ p & = \overline{a}ma.\overline{a}mb.a(l_{(xy)}).L(l_{xy}) \mid \overline{a}bc.p' & \qquad L(l_{xy}) = l_{xy}.L(l_{xy}) \end{array}$$

M receives two sites at m, creates a new link between them and sends this link from m to the first of the received sites. The process p has two components: the first component sends a and b from a to m, waits for a link at a and then activates a (persistent) transportation service over this link, which can be used by the other component; the second component sends c from a to b. The process q simply waits for a datum at b. Finally, the process L repeatedly activates a transportation service over its argument: this is necessary, because transportation services can only be used once. The whole system is $S = p \mid M \mid q \mid L(l_{am}) \mid L(l'_{ma})$, where l_{am} and l'_{ma} are the links that p and M use to interact.

We have that p, $L(l_{am})$ and M can do the following transitions

$$p \xrightarrow{\bullet;\overline{a}ma} \overline{a}mb.a(l_{(xy)}).L(l_{xy}) \mid \overline{a}bc.p' \qquad L(l_{am}) \xrightarrow{a;l_{am};m} L(l_{am}) \qquad M \xrightarrow{mma;\bullet} m(y).(l_{ay})\overline{m}al_{ay}.M$$

where $\bullet;\overline{a}ma$ represents the beginning of transmission as a path of length zero, analogous to the π-calculus output action: the $\bullet$ on the left side indicates that the path can only extend rightward, i.e. subsequent hops will be listed after $\bullet$ from left to right in the form of a sequence of links; the string $\overline{a}ma$ describes the path, telling (from left to right) the site where the datum is available, the destination site and the datum. Symmetrically, $mma;\bullet$ means that a, which has destination m, is received at m and then goes through a path of length zero; it is analogous to the π-calculus input action. The label $a; l_{am}; m$ represents the activation of a transportation service over l_{am}.

When these processes are put in parallel in S, the above paths can be concatenated, resulting in a path that represents a complete transmission over l_{am}

$$S \xrightarrow{\bullet;l_{am};\bullet} \overline{a}mb.a(l_{(xy)}).L(l_{xy}) \mid \overline{a}bc.p' \mid m(y).(l_{ay})(\overline{m}al_{ay}.M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) \ .$$

As in the π-calculus, the transmitted datum, namely a, is not observable. Then, a sequence of possible transitions after this one is:

$$\begin{array}{ll}
\cdots \xrightarrow{\bullet;l_{am};\bullet} a(l_{(xy)}).L(l_{xy}) \mid \overline{a}bc.p' \mid (l_{ab})(\overline{m}al_{ab}.M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) & \text{(transmission of } b\text{)} \\
\xrightarrow{\bullet;l'_{ma};\bullet} (l_{ab})(L(l_{ab}) \mid \overline{a}bc.p' \mid M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) & (l_{ab} \text{ scope extension, } l_{ab} \notin fn(p')) \\
\xrightarrow{\bullet;\bullet} (l_{ab})(L(l_{ab}) \mid p' \mid M) \mid q'[c/x] \mid L(l_{am}) \mid L(l'_{ma}) & \text{(transmission of } c\text{)}
\end{array}$$

Notice that the last transition hides the link used for transmission, namely l_{ab}, because it is restricted. We just observe $\bullet;\bullet$, analogous to the π-calculus τ-action.

The concurrent semantics allows observing in parallel all the pieces of a path. We may e.g. observe S doing $\bullet;\overline{a}ma \mid a; l_{am}; m \mid mma;\bullet$, which represents a three-element multiset. These observations are those making the behavioral equivalence on the concurrent semantics finer and compositional.

The major difference between NCPi and SCEL is the level of abstraction: SCEL is a parametric language, where many details are left unspecified, while NCPi focuses on a specific aspect of distributed systems, i.e. their connectivity. Other aspects are compared in the following.

Communication paradigm. The SCEL core language adopts an asynchronous mechanism, even if some level of synchrony can be recovered through interaction policies). On the contrary NCPi, being based on the π-calculus, is synchronous, in the sense that communication involves a sender, a receiver and transportation services providers, and the result of the synchronization of these processes is observed as a single transition. However, in NCPi there is room for asynchronous variations. For instance, we can think of a fully asynchronous version, based on the asynchronous π-calculus, where the semantics shows single forwarding steps. Adapting the SOS rules is straightforward

$$\frac{}{\overline{a}br \xrightarrow{\bullet;\overline{a}br} \mathbf{0}} \qquad \frac{p \xrightarrow{\bullet;\overline{a}br} p' \quad q \xrightarrow{a;l_{ac};c} q'}{p \mid q \xrightarrow{\bullet;l_{ac};\bullet} \overline{c}br \mid p' \mid q'} \qquad \frac{p \xrightarrow{\bullet;\overline{a}ar} p' \quad q \xrightarrow{aar;\bullet} q'}{p \mid q \xrightarrow{\bullet;\bullet} p' \mid q'}$$

Shifting from synchronous to asynchronous communication might have an impact on the theory, as happens for the π-calculus. A further investigation is this direction has yet to be carried out.

Resources and their categorical model. In NCPi, the resources of a process are its networks nodes and connections, corresponding to its free names. Modeling connections as names enables restriction and extrusion to generate new network pieces. The network owned by a NCPi process can be seen as

a graph, so (some subcategory of) the category of graphs is a candidate index category for presheaves. The SCEL core language does not provide resource management functionalities, but we can imagine that these will be part of some of its dialects, in the form of suitable policies and knowledge handling mechanisms. These languages will then admit categorical models.

A network of robot communications Besides our motivating example, we are exploiting the use of NCPi for modeling communication between robots. In fact, robots in an ensemble can be regarded as nodes of an ad-hoc network, where two robots are connected if they are able to communicate directly with each other. Communication to robots outside the signal range involves a chain of forwarding between robots until the destination is reached. Translating to NCPi, we could have a process for each robot, with the communication device being modeled as a subprocess activating transportation services towards each other robot within the signal range.

Conclusions and on-going work We provided motivations for investigating a general framework for resource-aware calculi. In particular, we considered presheaf models, that are algebraic/coalgebraic categorical models where the structure of resources is decoupled from the way resources are used. Then we presented NCPi, a network-aware extension of the π-calculus whose syntax allows expressing the creation and the activation of connections, and whose semantics shows multisets of routing paths that are covered concurrently. Finally, we described how NCPi relates to SCEL.

We plan to devise an asynchronous version and a presheaf model for NCPi, and investigate a general preasheaf-based framework covering various kinds of resources. Finally, we want to apply this framework to SCEL, as soon as resource management mechanisms will be explicitly specified.

2.2.2 State to function LTSs for stochastic process calculi (Task 2.1)

Several, significantly different, approaches have been proposed for handling transition multiplicity correctly. In [DLLM09c], we proposed a variant of LTSs, namely *Rate Transition Systems* (RTSs), as a tool for providing a uniform semantics to some of the most representative SPCs. In RTSs, a transition is a triple of the form $(P, \alpha, \mathscr{P})$. The first and second components are as in LTSs, while the third component $\mathscr{P}$ is the *continuation function* which associates a real non negative value to each state P'. A non-zero value represents the rate of the exponential distribution characterizing the time for the execution of the action represented by α, necessary to reach P' from P via the transition. If, on the other hand, $\mathscr{P}$ maps P' to 0, then state P' is not reachable from P via the transition.

To provide a uniform general account of the many SPCs proposed in the literature, we propose *State to Function Labeled Transition Systems*, FuTSs for short, a natural generalization of RTSs. As mentioned above, in RTSs, the co-domain of continuation functions is required to be the set of non-negative real numbers, used as rates (or to represent non-reachability). In FuTSs the above constraint is removed: the co-domains of the continuation functions are only required to be *commutative semi-rings*, so that FuTSs are a *generic* framework. Let $\mathbb{C}$ stand for a commutative semi-ring and $\mathbf{FTF}(S, \mathbb{C})$ denote the class of *total* functions from set S to $\mathbb{C}$ with finite support. In the simplest case, an *L-labeled* FuTS *over* $\mathbb{C}$ is a tuple $(S, L, \mathbb{C}, \rightarrowtail)$ where S is a countable, non-empty, set of *states*, L is a countable, non-empty, set of transition *labels*, $\mathbb{C}$ is a commutative semi-ring, and $\rightarrowtail \subseteq S \times L \times \mathbf{FTF}(S, \mathbb{C})$ is the *transition relation*. It is easy to see that standard CTMCs are isomorphic with total deterministic $\{\delta^e\}$-labeled FuTSs over $\mathbb{R}_{\geq 0}$, where the label δ^e denotes exponentially distributed random delay. Similarly, by conventionally using the label π for denoting discrete random experiments, DTMCs are isomorphic with total deterministic $\{\pi\}$-labeled FuTSs over $[\mathbf{0}, \mathbf{1}]$, where $[\mathbf{0}, \mathbf{1}]$ is an appropriate commutative semi-ring built on the $[0, 1]$ interval and every continuation $\mathscr{P}$ is a probability distribution function, i.e. $\oplus \mathscr{P} = 1$. RTSs coincide with total deterministic $\Delta_{\mathcal{A}}$-labeled FuTSs over $\mathbb{R}_{\geq 0}$, for given action

set $\mathcal{A}$, with $\Delta_{\mathcal{A}} =_{\text{def}} \{\delta^e_a \mid a \in \mathcal{A}\}$ (where δ^e_a means that action a has exponentially distributed duration). Finally, the model of Interactive Markov Chains [Her02], where non-deterministic behavior is integrated with the stochastic one, is isomorphic with total deterministic $(\{\delta^e\} \cup \mathcal{A})$-labeled FuTS over $\{\mathbb{R}_{\geq 0}, \mathbb{B}\}$, where, of course, we assume $\delta^e \notin \mathcal{A}$—in this last case we used the general definition of FuTS, which allows the use of finite *families* of commutative semi-rings.

A rich set of operators on $\mathbf{FTF}(S, \mathbb{C})$ is available, such as pointwise definition $[s_1 \mapsto \gamma_1, \ldots, s_m \mapsto \gamma_m]_{\mathbb{C}}$, addition $(\mathscr{P} + \mathscr{Q})\, s =_{\text{def}} (\mathscr{P}\, s) +_{\mathbb{C}} (\mathscr{Q}\, s)$, summation $\oplus \mathscr{P} =_{\text{def}} \sum_{s \in S} (\mathscr{P} s)$ and several multiplications. The definition of the FuTS operational semantics of SPCs are heavily based on $\mathbf{FTF}(S, \mathbb{C})$-operators, in an SOS-like style, making the framework simple, powerful and elegant.

Conclusions and on-going work A uniform framework for the definition of Stochastic Process Calculi [HHK02] has been introduced, as detailed in [DLLM11], while earlier work of the authors can be found in [DLLM09a, DLLM09b, DLLM09c]. The framework is based on the notion of *State to Function Labeled Transition Systems* and allows for an SOS-like language definition, providing a simple and elegant solution to several issues typical of this class of languages, e.g. transition multiplicity in relation with the rate condition principle of Continuous Time Markov Chains.

We plan to apply the State to Function Transition Systems framework to the definition of quantitative extensions of the ASCENS language, SCEL, with particular emphasis on stochastically timed dialects. Furthermore, we are interested in investigating on a co-algebraic view of FuTSs.

3 On Task 2.2: Adaptive SCs: building emergent behavior from local/global knowledge

In the Introduction we wrote that the research on this task should be concerned with the "develop[ment of] robust mathematical foundations for interaction scenarios [...] address[ing] models that can favor a mixture of static and dynamic analysis tools by adhering to the NCE schema".

At the end of the first year, also the work on this task shows two different facets. One concerns the development of models for interaction scenarios that adhere to the NCE schema, and it focused on *Contract distillation by distributed synthesis*. On a more foundational issue, the identification of *Conceptual models for autonomicity* was planned, and two frameworks for different views of adaptivity were proposed.

3.1 Task 2.2: strand on Contract distillation by distributed synthesis

This facet of the task, reported in §3.1.1, addresses client-service interactions distinguishing between three phases: Negotiate, Commit and Execute. The participants negotiate their behaviors, and if an agreement is reached they commit and start an execution which is guaranteed to respect the interaction scheme agreed upon. These ideas are materialized via a calculus of contracts enriched with semiring-based constraints [BCDM11], which allow clients to choose services and to interact safely with them. A concrete representation of these constraints with logic programs is straightforward, reducing constraint solution (and thus the establishment of a contract) to the execution of a logic program.

3.1.1 Constraints for service contracts (Task 2.2)

Interaction scenarios that are characterized by highly dynamic, autonomic components are not easily verifiable, because of the difficulty of correctly designing communication protocols. Very successful has been the decision to abstract away from the actual data and their algorithmic complexity, and to focus on communication properties. Typically, the number of states of a system becomes finite (while

possibly very large) and verification techniques based on model checking and static analysis become feasible, and quite effective. The typical property to check is lack of deadlock, more precisely, stuck-freedom [RR02]: there are no messages waiting forever to be sent or sent messages which are never received, thus assuring that the interaction between partners will successfully end. The key idea is to associate to a process an abstract description of its behavior, and to check if the pairs of processes which are expected to communicate do match. The most common styles used to model abstract process behaviors are session types (originally introduced in [THK94]) and behavioral contracts [CGP09]. The extended version of [CDCP11] discusses several different approaches to the global and local descriptions of communication protocols. We criticize the existing approaches on two grounds. The first problem is that the result of the static analysis is on-off: either the interaction is acceptable or not. The second problem is that the verification process must be repeated for every pair of partners. It would be more convenient to split it into two parts: a *compilation* step, to be executed at deployment time, where the abstract behavior is determined, and a *matching* step to be executed at run time, for every pair of partners willing to communicate. We believe that an approach based on constraints would be able to overcome these difficulties: (i) the compilation step should generate a constraint modeling the behavior; (ii) the matching step should be simply constraint composition, successful only if the resulting constraint is satisfiable; (iii) the actual execution should be monitored by ask-like guards, reminiscent of concurrent constraint programming. The advantage of a constraint-based approach [BCDM11] is clear: the necessary constraints can be built inductively at compile time, composed at matching time and tested at run time taking advantage of concepts well-studied in the area of constraint programming. We present our constraint system as an instantiation of the class of *named constraint semirings*, which have been originally proposed as the underlying structure of the cc-pi calculus [BM07], a process calculus for modeling agreements on non-functional parameters in a service oriented scenario. The target calculus we propose is close in spirit to the cc-pi calculus, except for the fact that the primitives of our target calculus are meant to model two-party interactions.

Constraint semirings [BMR97] are semirings with an idempotent additive operation and a commutative multiplicative operation. Named constraint semirings in addition come equipped with a notion of relevant names that allows plugging constraints into languages with an explicit concept of names. Named constraint semirings inherit from ordinary constraints on the boolean semiring properties and efficient algorithms, like constraint propagation and dynamic programming. We exploit a simple and standard named constraint semiring: the one employed by logic programming, where the Herbrand signature contains as many unary operations as actions and a constant to model termination. The semiring values are sets of assignments with Herbrand terms to all the names. However, only the assignments to the tuples of names forming the support are relevant. The correspondence with logic programming is quite simple: given a set of clauses and a goal $P(x_1, \ldots, x_n)$, its semantics in terms of our constraint system is the set of all the ground instantiations of the goal which satisfy the clauses. The support is at most the set $\{x_1, \ldots, x_n\}$. Goal composition is multiplication, while multiple clauses for the same goal model sum. Variables appearing in the body but not in the head of a clause are existentially quantified. The effect of recursive clauses is obtained by an explicit fixpoint operator.

Our approach [BCDM11] combines a mixture of static and dynamic analysis tools by adhering to the Negotiate-Execute-Commit (NCE) scheme. Agents negotiate certain desired behaviors, but without any guarantee of success. However, if and when an agreement is reached (commit), under certain conditions a coordinated computation of the involved agents can start, which is guaranteed to have the properties agreed upon in the negotiation phase. Specifically, we present a simple source calculus with client and service processes and with a semantics given in terms of labeled transition systems: the clients are recursive and can place nested service calls, while services are permanent (namely a service is not consumed by a call) and nonrecursive. The calculus is nondeterministic, with external choices, and a client-service behavior which allows for more choices is considered to be more

desiderable, provided they are stuck-free. A target calculus is then defined, where clients and services are compiled to, augmenting them with named constraint semirings, which encode their behavior.

To understand our approach, let us first consider the case of a client without nested calls and with a single service. The source client is then compiled into a target client combined with a constraint with just one name in its support, say x. The constraint is thus just a set of traces, representing the behavior of the client. The compiled code is very similar to the source, except that its choices are guarded by *check* constructs, similar to the *ask* constructs of concurrent constraint programming, which enable the corresponding continuations only if the global constraint allows it. The source service is also compiled, yielding a constraint on y which represents its behavior, but no check guards are included. The negotiation phase consists of multiplying the two constraints, and their result with the constraint $x = y$. The resulting constraint contains exactly all the executions of the source client-service system which are not stuck. If the constraint is not 0, i.e. if it is not the empty set, the commit takes place, and the execution phase can start. Thanks to the check guards, the traces possible for the target client-service system are exactly those in the constraint. Notice that while the client (service) compilation is static, and thus it does not fit in the NCE scheme, it does not depend on the particular service (client) partner it will be matched to. Thus the open endness requirement is here satisfied by the possibility of deploying new services (clients) without the need of any further modification of the existing services (clients). For instance, let T be a client which offers the choice between co-actions $\overline{\alpha}$ and $\overline{\beta}$, and S be a service which offers the choice between actions α and γ, and only co-actions and actions interact successfully. Therefore T and S can safely interact only choosing $\overline{\alpha}$ and α, respectively. In our calculus we have $T = \overline{\square}.\overline{\alpha}.\overline{\boxtimes}.\mathbf{0}+\overline{\beta}.\overline{\boxtimes}.\mathbf{0}$ and $S = \square.\alpha.\boxtimes + \gamma.\boxtimes$, where $\overline{\square}$ is service call, $\overline{\boxtimes}$ is call end, and $\square$, $\boxtimes$ are service and end acceptance. The interactions offered by T and S are represented, respectively, by the constraints $c = (x = \alpha(end)) \oplus (x = \beta(end))$ and $d = (y = \alpha(end)) \oplus (y = \gamma(end))$ (noting that we write constraints using only actions). It holds that $c \otimes d \otimes (x = y) = (x = \alpha(end)) \otimes (y = \alpha(end)) \otimes (x = y) \neq 0$, which reflects the fact that the only successful interaction between T and S is (over) α.

Let us now consider the general case of a client with nested calls and several services. Services are compiled in the usual way. The behavior of the client, instead, must be represented by a constraint with several names in the support. In fact, different service calls may not be independent: imagine that the client makes a choice in an inner call which must be matched by the corresponding service. Then the client returns to the outer level and makes another choice which must be matched this time by the service corresponding to the outer call. The two choices may be dependent, and this requirement is represented by a constraint with a two-name support. Thus the ability of the constraint system of representing sets of tuples of traces allows us to guarantee stuck-freedom for complex client-service pairs, which at the best of our knowledge have not been considered in the literature by now. An example of this kind of clients is $\overline{\square}.\overline{\alpha}.\overline{\square}.\overline{\beta}.(\overline{\gamma}.\overline{\boxtimes}.\overline{\delta}.\overline{\boxtimes}.\mathbf{0} + \overline{\mu}.\overline{\boxtimes}.\overline{\rho}.\overline{\boxtimes}.\mathbf{0})$, where the choices made by the two nested calls depend on each other. Such a dependency can arise, for instance, when modeling a traveler who asks both for a flight ($\overline{\alpha}$) to an airline company and for a room ($\overline{\beta}$) to a hotel in two alternative different dates. The client request (room, flight) are represented respectively by actions $\overline{\gamma}$, $\overline{\delta}$ for one date and $\overline{\mu}$, $\overline{\rho}$ for the other date. This client offers then the choice between the pairs of traces $< \alpha(\delta(end)), \beta(\gamma(end)) >$ and $< \alpha(\rho(end)), \beta(\mu(end)) >$. The constraint resulting from the negotiation,

$$(x_1 = \alpha(\delta(end))) \otimes (x_2 = \beta(\gamma(end))) \oplus (x_1 = \alpha(\rho(end))) \otimes (x_2 = \beta(\mu(end))),$$

with support $\{x_1, x_2\}$, obliges the run of related service call to be coherent.

Conclusions and on-going work We proposed a NCE approach for guaranteeing stuck-freedom in interactions between client and services [BCDM11]. We have augmented a client-service calculus with semiring-based constraints, which allow clients to choose services and to interact with them in a safe way. A run time combination (multiplication in the simple cases) of client and service constraints ensures that all and only the stuck-free interactions are possible.

It would be interesting to generalize the current target calculus by exploiting the formalism of Soft Constraint Logic Programming [BMR01]. In that paper, the ground semantics of a logic program (a goal and set of clauses) is not a set of ground assignments of the free variables of its goal, but rather a function from ground assignments to values of (another) constraint semiring. These values could give a measure of how acceptable the assignments are. Such functions, computed point-wise, form again a constraint semiring, and thus the formal treatment turns out simple and elegant. In particular, the three semantics of logic programming (operational, denotational and model theoretical) can be defined also for soft constraint logic programming and proved equivalent. In our setting, a particular client-service computation would not be only possible or impossible, but it could be assigned an acceptance weight, which might itself be structured by measuring the quality of service obtained in the interaction.

In a different direction, we plan to study how to integrate our semiring-based constraints and, more in general, to exploit the interaction mechanism of the concurrent constraint pi-calculus [BM07] within the ASCENS language SCEL. We expect the most natural way would be by representing the SCEL knowledge in terms of constraints while the SCEL policies by (a variant of) the communication mechanism of the concurrent constraint pi-calculus which allows merging constraints arising from different SCs. In this respect, the main challenge seems to be how to handle in SCEL names which are shared by different SCs or even SCEs.

3.2 Task 2.2: strand on Conceptual models for autonomicity

While the meaning of "adaptivity" may seem intuitively obvious, various communities use different and incompatible definitions of the term, and surprisingly few generally applicable, precise definitions are available in the literature.

For the second facet of the task we therefore present two conceptual frameworks for adaptivity: The first one, presented in §3.2.1 [HW11], is concerned with "black-box" adaptation and based on the GEM system model for ensembles. Black-box adaptation describes how well a system can perform in various environments without looking at the internal mechanisms used to achieve adaptation. It gives rise to a preorder of adaptivity on ensembles based on their ability to satisfy goals or maximize a performance measure in different environments. The second framework, presented in §3.2.2 [BCG+11], defines a notion of "white-box" adptation conceived around a prominent role of clearly and neatly identified *control data*: computational data that govern the execution and are conveniently managed to enact adaptive behaviors. White-box adaptation is therefore concerned with how the adaptation process is achieved, not with the environments to which the system can adapt. Most foundational models and programming paradigms proposed for describing or implementing autonomic or adaptive computation fit perfectly within these two frameworks.

3.2.1 GEM: A General Model for Ensembles and Black-Box Adaptation (Task 2.2)

The goal of the *General Ensemble Model (GEM)* is to provide a denotational model of ensembles and to define notions such as (black-box) adaptation, awareness and emergence based on a solid mathematical basis. GEM views systems as relations, similar to Mesarovic and Takahara's General Systems Theory [MT75], but GEM allows the specification of ensemble properties using various logics, and it also introduces heterostatic (i.e., optimizing a utility function) and probabilistic systems.

If I is a (finite or infinite) set, and $\mathcal{V} = (V_i)_{i \in I}$ a family of sets we define an *ensemble* or *(general) system* or *component* of type $\mathcal{V}$ as a relation S of type $\mathcal{V}$. It is in many cases helpful to regard some of the sets $V_i \in \mathcal{V}$ as inputs and others as outputs, even though these are often not system-inherent properties. We can do so by defining an isomorphism which, roughly speaking, divides $\mathcal{V}$ into inputs X, outputs Y and internal state Z, and identifies S with a relation of type (X, Y, Z), see Fig. 3.

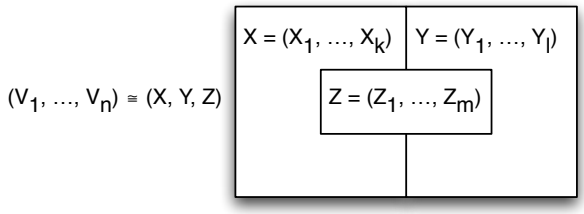


Figure 3: Ensemble in GEM

Since we are usually interested in systems that change their behavior over time, we define modal and timed systems: Let T, the *possible worlds* or *time structure*, be a set, let R be a binary relation on T, $(A_i)_{i\in I}$ a family of sets, and let $V_i = \mathcal{F}[T \to A_i]$. A *modal system* or *modal ensemble over* $(A_i)_{i\in I}$ *with possible worlds* T is a general system S over $(V_i)_{i\in I}$. If R is a preorder we call T a *time system* or *time ensemble over* $(A_i)_{i\in I}$ *with time structure* T. The available space does not allow us to introduce heterostatic or probabilistic ensembles, but GEM can easily accomodate these extensions.

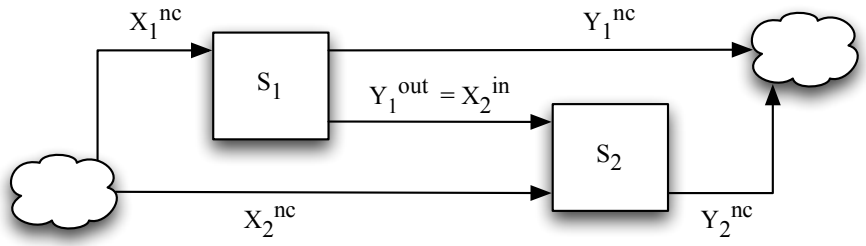


Figure 4: Combination operator for partial cascade

The hierarchical construction of most ensembles is mirrored in GEM by *combination operators* that combine several relations (regarded as models of components) into a new relation (the model of the system composed of these components). By defining appropriate combination operators, we can express any kind of composition of components. Fig. 4 depicts a partial cascade, a simple combination operator that connects some of the outputs of a system S_1 to corresponding inputs of S_2 while leaving other inputs and output connected to the environment.

In order to specify properties of systems we connect the relational system model to different logics: If $\mathfrak{L}$ is a suitable logic and γ a formula of $\mathfrak{L}$ that represents a goal or requirement of the system, we define the notion "System S satisfies goal γ", written $S \models \gamma$. This very general construction allows us to express properties of ensembles in a wide variety of logics and therefore to apply many different formal methods to the investigation of ensembles expressed in GEM.

Having introduced the fundamentals of GEM we can now define black-box adaptation. To this end we assume that a combination operator $\otimes$ is given; this operator combines three systems (which may themselves be composed of simpler systems): an environment η which we regard as mostly outside the control of the system designer, a "network" or sensor/actuator system ν which simulates or represents the connection of the ensemble to the environment, and the system S. We assume that S is designed by the software developer and that it has to achieve a certain goal in the given environment or in a range of environments in order to be considered successful.

Consider a system S which is executed in environment η with network ν. In order to realize the

given specification or goal γ the system has to satisfy following properties:

$$\eta, \nu, S \not\models \bot \tag{1}$$
$$\eta, \nu, S \models \gamma \tag{2}$$

Property (1) states that the system resulting from the composition of S, η and ν is not empty. This condition is necessary since an inconsistent systems vacuously satisfies any goal γ. Property (2) immediately expresses the fact that the system satisfies the goal in question.

Usually we do not speak of adaptation when a system works in just a single, deterministic environment; we expect an adaptive system to work in a variety of different situations. Since environments are relations, this can be achieved by having an environment with non-deterministic and time-variant behavior. However it often simplifies the comparison and analysis of different systems if we do not merge different behaviors into a single environment and instead model adaptation by having several environments, e.g., η and η' such that the system satisfies the goal for all environments:

$$\eta, \nu, S \models \gamma \qquad \text{and} \qquad \eta', \nu, S \models \gamma$$

Different network conditions may be modeled in a similar way.

Adaptation to a new environment or network is not the only possible kind of black-box adaptation; it might also be necessary to change the goals that an ensemble pursues while leaving the environment constant, or, in other words, the system may have to adapt to new requirements. In that case we obtain the adaptation condition

$$\eta, \nu, S \models \gamma'$$

If the new goal γ' is implied by the old goal γ, any system satisfying γ already satisfies γ'. Otherwise it is, in general, not possible for a system to adapt to new goals unless this goal is communicated to the system, either by one of the unconnected inputs of the system or, more frequently, by changes in the environment. For example, we might change the color of a beacon from green to red to signal to a foraging robot that we want it to stop foraging and return to the base station. Therefore, it is usually not sensible to request that a system can adapt to any change in environment and goals, we rather have to restrict adaptation to these scenarios where the goal is correctly communicated to the ensemble.

To formalize these notions we define an *adaptation domain* $\mathcal{A}$ that describes a range of environments $\mathcal{E}$, networks $\mathcal{N}$ and goals $\mathcal{G}$, to which we want the system to adapt and define the notion S *can adapt to* $\mathcal{A}$, written $S \Vdash \mathcal{A}$:

$$\mathcal{A} \subseteq \mathcal{E} \times \mathcal{N} \times \mathcal{G}$$
$$S \Vdash \mathcal{A} \iff \forall (\eta, \nu, \gamma) \in \mathcal{A} : \eta, \nu, S \models \gamma$$

The *adaptation space* $\mathfrak{A}$ is a set of adaptation domains, $\mathfrak{A} \subseteq \mathfrak{P}(\mathcal{E} \times \mathcal{N} \times \mathcal{G})$. It is partially ordered by inclusion; for any adaptation space we define a preorder of adaptivity for systems as follows:

$$S \sqsubseteq S' \iff \forall \mathcal{A} \in \mathfrak{A} : S \Vdash \mathcal{A} \implies S' \Vdash \mathcal{A}$$

In that case we say that S' is *at least as adaptive as* S (with respect to $\mathfrak{A}$). If $S \sqsubseteq S'$ and there is an adaptation domain $\mathcal{A} \in \mathfrak{A}$ for which $S' \Vdash \mathcal{A}$ but $S \not\Vdash \mathcal{A}$ then we say that S is *less adaptive than* S' or that S' is *more adaptive than* S (with respect to $\mathfrak{A}$) and write $S \sqsubset S'$.

Black-box adaptation defined in terms of adaptation spaces gives us a very general notion to compare the capability of various ensembles to adapt to changing environments. It does, however, not specify any mechanism how this adaptation can be achieved. To this end the notion of white-box adaptation, that will be introduced in the next section, is necessary.

3.2.2 White-Box Adaptivity: A conceptual framework (Task 2.2)

The aim of this task is to *develop robust mathematical foundations for interaction scenarios that are characterized by highly dynamic, autonomic components, that can update their behavior depending on the current environment, join and leave an interaction, fork new components, react to events and compensate past activities.* This is a challenging goal, that we address starting with some much easier questions that concern individual autonomic components rather than ensembles: *"When is a software system adaptive?"*, and *"how can we identify the adaptation logic in an adaptive system?"* We think that the limited effort placed in the investigation of the foundations of (self-)adaptive software systems might be due to the fact that it is not clear what are the characterizing features that distinguish such systems from plain ("non-adaptive") ones.

We are developing a conceptual framework for white-box adaptation [BCG+11], proposing a simple structural criterion to characterize adaptivity. Our framework requires to make explicit that the behavior of a component depends on some well identified *control data*. Now, we define *white-box adaptation* as the *run-time modification of the control data*. A component is thus deemed *adaptable* if it has a clearly identified collection of control data that can be modified at run-time. And if the control data are not identified or cannot be modified, then the system is not adaptable. Further, a component is *adaptive* if it is adaptable and its control data are modified at run-time, at least in some of its executions. And a component is *self-adaptive* if it is able to modify its own control data at run-time.

Under this perspective any computational model or programming language can be used to implement an adaptive system, just by identifying the part of the data that governs the behavior. Consequently, the nature of control data can vary considerably, ranging from simple configuration parameters to a complete representation of the program in execution that can be modified at run-time. This latter scenario is typical of computational models that support meta-programming or reflective features even if, at least in principle, it is possible for any Turing-complete formalism.

Starting from the MAPE-K model [Hor01], several contributions have described possible architectures for adaptive systems (or for *autonomic systems*, for which self-adaptivity is a main feature). According to the MAPE-K architecture, which is a widely accepted reference model, a self-adaptive system is made of a component implementing the application logic, equipped with a control loop that monitors the execution through sensors, analyses the collected data, plans an adaptation strategy, and finally executes the adaptation of the managed component through effectors; all the phases of the control loop access a shared knowledge repository. Adaptation according to this model naturally fits in our framework with an obvious choice for the control data: these are the data of the managed component which are either sensed by the monitor or modified by the execute phase of the control loop. Thus the control data represent the interface exposed by the managed components through which the control loop can operate, as shown in Fig. 5. Clearly, by our definitions the managed component is adaptive, and the system made of both the component and the control loop is self-adaptive.

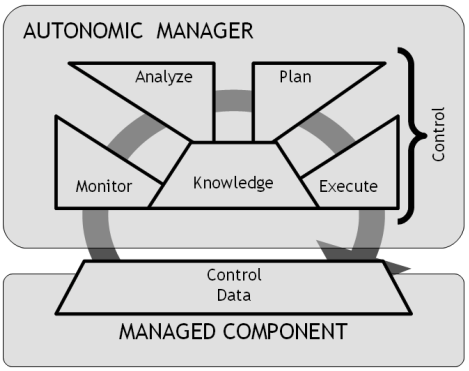


Figure 5: Control data in MAPE-K.

The construction can be iterated, as the control loop itself could be adaptive. Think e.g. of an adaptive component which follows a plan to perform some tasks. This component might have a manager which devises new plans according to changes in the context or in the component's goals. But this planning component might itself be adaptive, where some component controls and adapts its planning strategy, for instance determining the new strategy on the basis of a tradeoff between

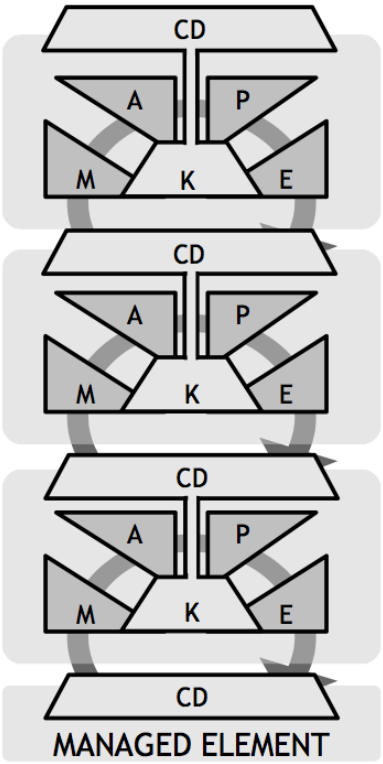


Figure 6: Tower of adaptation.

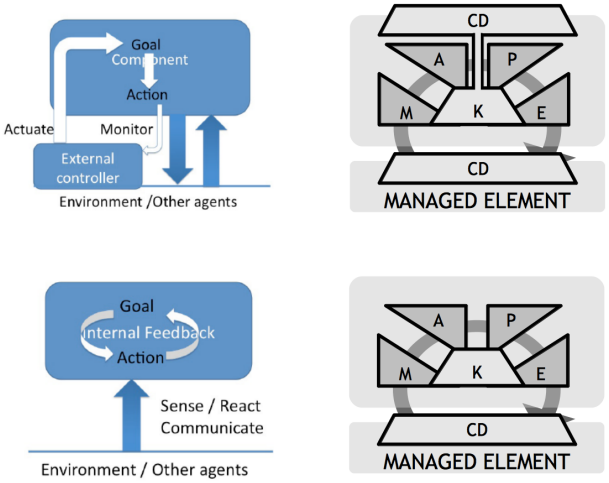


Figure 7: External (top) and internal (bottom) patterns.

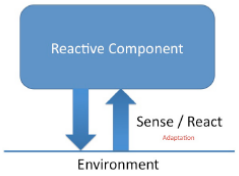


Figure 8: Reactive pattern.

optimality of the plans and computational cost. In this case also the manager (the control loop) should expose in an interface its control data, which are conceptually part of its knowledge repository. In this way, the approach becomes compositional in a hierarchical way, which allows one to build towers of adaptive components (Fig. 6).

Patterns from the taxonomy of [CPZ11] can be cast easily in our framework (see Fig. 7) as well. In the internal control loop pattern, the manager is a wrapper for the managed component and it is not adaptable, while in the external control loop pattern the manager is an adaptable component that is connected with the managed component. The taxonomy of [CPZ11] includes a third adaptive pattern that describes *reactive* components (see Fig. 8). Such components are capable to modify their behavior in reaction to an external event, without any internal control loop. In our conceptual framework, a reactive system of this kind is (self-)adaptive if we consider as control data the variables that are modified by sensing the environment.

The nature of control data can vary considerably depending both on the degree of adaptivity of the system and on the nature of the computational formalisms used to implement it. Examples of control data include configuration variables, rules (in rule-based programming), contexts (in context-oriented programming), interactions (in connector-centered approaches), policies (in policy-driven languages), aspects (in aspect-oriented languages), monads and effects (in functional languages), and even entire programs (in models of computation exhibiting higher-order or reflective features).

We started developing a simple formal model, based on labeled transition systems, as a proof-of-concept for validating the idea of developing formal models of adaptive systems where the key features of our approach are first-class citizens. We applied it to model a small scenario of an archetypal adaptive system, namely a robot swarm whose goal is to collect items in an unknown area.

Conclusions and on-going work We presented a conceptual framework for adaptivity [BCG+11]. Its key feature is a neat identification of the *control data* governing and enacting adaptive behaviors.

Most adaptive behaviors approaches either based on foundational models (such as logical reflection) or programming paradigms (such as context-oriented programming) fit perfectly in the framework.

We plan to exploit our conceptual framework by developing sound design principles for architectural styles and patterns in order to ensure correctness-by-design, and guidelines for the development of adaptive systems conforming to such patterns. We also plan to develop analysis and verification techniques for adaptive systems grounded on the central role of control data. For example, data- and control-flow analysis techniques could be used to separate, if possible, the adaptation logic of a system (that modifies the control data) from the application logic (that just reads them). Another current line of research aims at developing variants of computational models specifically inspired on our framework. Some steps in this direction have been already taken, starting from a very basic concept of adaptive transitions systems [BCG+11]. Another candidate is the reflective, rule-based approach from [MT02]: we plan to use the Maude framework to develop prototype models of archetypal and newly emerging adaptive scenarios. Even if we focused the present work on adaptation issues of individual components, we also intend to develop a framework for adaptation of *ensembles*, i.e., massively parallel and distributed autonomic systems which act as a sort of swarm with emerging behavior. This could require to extend our *local* notion of control data to a *global* notion, where the control data of the individual components of an ensemble are treated as a whole, which will possibly require some mechanism to amalgamate them for the manager, and to project them backwards to the components.

4 On Task 2.3: Modeling SCEs with collaborative and competitive behavior

The research on this task has the ambition of "develop[ing] a theory combining as much as possible the flexibility and compositionality of computer science semantic models with the expressiveness of models developed by economic theorists", with an in-depth analysis of the "adapt[ion] and re-use in this context many co-algebraic tools well-known for ordinary transition systems".

One side concerns the application of *Game paradigms for service composition*, adopting either non-cooperative job scheduling games, with an interest in abstractly capturing those scenarios where a Nash equilibrium is reached, and minority games for service composition, as needed by e.g. a peer-to-peer energy management scenario. On a more foundational issue, *Coalgebraic techniques for dynamical systems* pushed the coalgebraic view of weighted automata: non-deterministic automata where each transition has also a quantity expressing the cost of its execution.

4.1 Task 2.3: strand on Game paradigms for service composition

We formulate in §4.1.1 a repeated non-cooperative job scheduling game, proposed in [BMT10, BMT11], whose players are Grid sites and whose strategies are scheduling algorithms. We exploit the concept of Nash equilibrium to express a situation in which no player can gain any profit by unilaterally changing its strategy. Specifically, we investigate different strategies in different contexts and we show whether each such strategy is a Nash equilibrium or not. In the negative cases we provide counter-examples, in the positive cases we either give formal proofs or motivate our conjectures by experimental results supported by simulations and exhaustive search.

An fruitful application of the adaptation in service component is peer-to-peer energy management. The domestic energy market is changing with the increasing availability of energy generating home-devices, such as solar panels, which provide energy for the house where they are installed as well as enabling the selling of surplus energy. In §4.1.2 we consider a scenario where households have the chance to trade energy for purchasing to and for selling from a number of different actors.

4.1.1 A game-theoretic model of Grid systems (Task 2.3)

Grid Computing and Cloud Computing aim at creating an illusion of a simple and yet large virtual computer from a great set of heterogeneous computers sharing various resource types to benefit a virtual organization. A common scenario includes several sites which share their computational and storage resources in the form of clusters of machines and a global scheduler which is responsible for the scheduling of jobs over sites. Grid sites may have conflicting interests, for e.g. some site prefers first to execute its own local jobs over the Grid jobs, in order to minimize the sum of completion time of the former jobs. In such non-cooperative systems, since Grid sites are not under control of a centralized broker, optimization does not amount to maximizing/minimizing a unique common function, but rather to find a stable situation in which, for instance, sites guarantee to equally treat remote and local jobs. In order to analyze the impact of selfishness and the potential non-cooperativeness in Grid systems, in [BMT10, BMT11] we develop a theory combining the flexibility and compositionality of computer science semantic models with the expressiveness of the game-theoretic approaches.

Game theory [Osb04] attempts to formally capture behaviors in strategic situations, in which an individual agent success in making choices depends on the choices of others. A game consists of a set of players, a set of possible actions (moves), and a set of specified player payoffs that are assigned depending on the actions performed by every player. At each step, every player chooses an action and gets a payoff in return. A player strategy is a sequence of actions such that each action refers to a round of the game. A central notion in Game theory is that of Nash equilibrium [Nas51], a situation in which no player can improve its own payoff by unilaterally changing its strategy.

We formulate a game in which Grid sites are the players of the game, and the actions available to them range over various job scheduling policies. The game we define is *non-cooperative*, namely players take decisions independently from each other, and *repeated*, such that it consists in some number of repetitions of a base game, and we assume that the number of repetitions is finite but unknown to players. Each time there is a job to be executed in the Grid, each player bids or not for the purpose of winning the job. If a player is willing to execute a Grid job, it will propose its Earliest estimated Response Time (ERT) which is an estimation of the time interval between the job submission and the beginning of the job execution. Next, the global scheduler assigns to the winner - namely, the site that guarantees the minimum ERT for that job - (i) the execution of that job (ii) a payoff that is the length of the job. Conversely, a zero payoff is assigned to other sites. The aiming environment is defined by the availability/non-availability of Grid and local jobs, as well as by the workload on each one of their site.

The contribution of [BMT10] is two-fold: on the one side to formalize the notion of *Labeled Transition Game*, which basically allows formulating a non-cooperative game in terms of a kind of Labeled Transition Systems; on the other side, to formally prove that in absence of local jobs, the eager strategy profile where every site bids its ERT upon arrival of a new Grid job is a Nash equilibrium. Paper [BMT11] extends the analysis carried out in [BMT10] to different strategies. Specifically, we have provided a coherent classification by formally and/or experimentally proving a class of results which hold under different orthogonal Grid features. Such an analysis considers all the cases which can be obtained by varying three parameters: (i) presence/absence of local jobs, (ii) presence/absence of heavy load - the *heavy load* condition requires that at any time unit there must be enough incoming jobs so that every site that is free and willing to execute a Grid job cannot be inactive, (iii) variable/fixed job length during the whole repeated game, in the second option jobs must have the same length. Depending on the presence of local jobs, two different eager strategies can be adopted by the players: in the presence of both local and Grid jobs, the selfish strategy according to which a player bids for a Grid job only if there is no incoming local job; in absence of local jobs, the collaborative strategy which states that a player bids for a Grid job if there is any available. Each strategy is shown to be or not to be a Nash equilibrium upon varying job length and heavy load parameters.

Conclusions and on-going work We analyzed the potential non-cooperativeness and the impact of selfishness in Grid systems, by exploiting the concept of Nash Equilibrium inherited from Game theory [BMT10]. We gave a coherent classification [BMT11] of use cases specifying different strategies under different orthogonal Grid features and stated whether each strategy is a Nash equilibrium or not.

We are confident that the approach discussed here could be adapted to Cloud Computing, with limited modifications. Indeed, Cloud basically inherits from Grid its underlying job scheduling mechanism. For this reason, we plan to apply our method to face some of the main challenges raised by the ASCENS Cloud case study.

4.1.2 Smart meter aware domestic energy trading agents (Task 2.3)

An interesting application of the adaptation in service component is represented by peer-to-peer energy management. In [CPCA11] we figure out a scenario where households will have the chance to trade energy for purchasing to and for selling from a number of different actors.

We exploited software agents as a suitable component-based technology to model and implement a system composed of different actors: consumers, which buy energy, prosumers, which produce and consume energy, gencos, which produce and sell energy on large scale. The scenario is complex because prosumers produce cheaper but more limited energy than gencos. Moreover, this scenario requires a high degree of adaptability, because of different aspects: (i) the need for energy is different during the day and varies during the week; (ii) weather conditions influence energy production, for instance a cloudy day decreases the energy produced by solar panels; (iii) the price of the energy produced by prosumers change depending not only on weather conditions but also on the base of the demand; (iv) gencos too have a threshold over which the energy becomes very expansive.

Given this scenario, we applied a game-theoretical approach. In particular, we applied an instance of the minority game, a family of games where the choice of the minority wins (in our case, the minority of consumers is expected to have a lower price). We implemented the system by Jade, and connected agents to a physical Smart Meter to provide a near-to-reality simulation. Results are quite good: the paid price is usually close to the expected price, and the whole production of energy is close to the needed one, avoiding waste of energy. This demonstrates not only that single agents are able to adapt to personal interests, but also that the emerging behavior is the one expected by the community.

Conclusions and on-going work The domestic energy market is changing with the increasing availability of energy micro-generating facilities. On the long run, households will have the possibility to trade energy for purchasing to and for selling from a number of different actors. We modeled such a futuristic scenario using software agents. We produced an implementation including the interfacing with a physical Smart Meter and provided simulation results. Given the high autonomy of the actors in the domestic market and the complex set of behaviors, the agent approach proves to be effective for both modeling and simulating purposes.

The simulation of agents managing an open energy market on the base of minority game theory pointed out the capability of the system to reach an equilibrium in a short period of time and, more important, based on choices that tend to the respect of the environment (see [CPCA11] for details).

4.2 Task 2.3: strand on Coalgebraic techniques for dynamical systems

Autonomic, and more generally adaptable and reconfigurable systems, are complex objects where such aspects like feedback and stability play a key role. Describing and analyzing those aspects calls for a systemic view of systems, where classical models of nondeterministic computation might be integrated with tools from Control Theory. In this context, *SCE*'s could actually be described and analyzed as dynamical systems. A suitable tool appears to be *weighted automata*, a generalization

of classical non-deterministic automata where each transition, in addition to an input letter, has also a quantity expressing the weight (e.g. cost or probability) of its execution, drawn from a semiring [KS86]. Recent investigation have pointed out that weighted automata are, in a very precise sense, a generalization of (discrete) time-invariant linear systems from Control Theory [Rut07, Bor09].

4.2.1 A coalgebraic view on Resource Aware operational models (Task 2.3)

The paper [BBB+11] is part of a line of an ongoing research that aims at re-using in the weighted automata context those co-algebraic tools well-known from ordinary transition systems. This will in turn lead to coinductive reasoning and algorithms for equivalence checking and minimization.

As for non-deterministic automata, the behavior of weighted automata can be expressed in terms of either (weighted) bisimilarity or (weighted) language equivalence. Coalgebras provide a categorical framework for the uniform study of state-based systems and their behaviors. We show that coalgebras can suitably model weighted automata in two different ways: coalgebras on **Set** (the category of sets and functions) characterize weighted bisimilarity, while coalgebras on **Vect** (the category of vector spaces and linear maps) characterize weighted language equivalence. Relying on the second characterization, we show three different procedures for computing weighted language equivalence. The first one consists in a generalization of the usual partition refinement algorithm for ordinary automata. The second one is the backward version of the first one, which leads to a matrix-based algorithm of cubic complexity. The third procedure relies on a syntactic manipulation of rational weighted languages seen as multivariate streams (a.k.a. formal power series).

Conclusions and on-going work Ultimately, one would like to develop a comprehensive coalgebraic framework integrating classical models of computation and Control Theory, where such concepts as feedback, controllability and stability could be described and analyzed.

References

[ABC+09] Farhad Arbab, Roberto Bruni, Dave Clarke, Ivan Lanese, and Ugo Montanari. Tiles for Reo. In Andrea Corradini and Ugo Montanari, editors, *WADT*, volume 5486 of *Lect. Notes Comp. Sci.*, pages 37–55. Springer, 2009.

[ADG98] Robert Allen, Rémi Douence, and David Garlan. Specifying and analyzing dynamic software architectures. In Egidio Astesiano, editor, *FASE*, volume 1382 of *Lect. Notes Comp. Sci.*, pages 21–37. Springer, 1998.

[Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[BBB+11] Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. Submitted, 2011.

[BBBS08] Ananda Basu, Philippe Bidinger, Marius Bozga, and Joseph Sifakis. Distributed semantics and implementation for systems with interaction and priority. In Kenji Suzuki, Teruo Higashino, Keiichi Yasumoto, and Khaled El-Fakih, editors, *FORTE*, volume 5048 of *Lect. Notes Comp. Sci.*, pages 116–133. Springer, 2008.

[BCDM11] Maria Grazia Buscemi, Mario Coppo, Mariangiola Dezani, and Ugo Montanari. Constraints for service contracts. In Roberto Bruni and Vladimiro Sassone, editors, *TGC*, volume in press of *Lect. Notes Comp. Sci.* Springer, 2011.

[BCDW04] Jeremy S. Bradbury, James R. Cordy, Jürgen Dingel, and Michel Wermelinger. A survey of self-management in dynamic software architecture specifications. In David Garlan, Jeff Kramer, and Alexander L. Wolf, editors, *WOSS*, pages 28–33. ACM, 2004.

[BCEH05] Paolo Baldan, Andrea Corradini, Hartmut Ehrig, and Reiko Heckel. Compositional semantics for open Petri nets based on deterministic processes. *Mathematical Structures in Computer Science*, 15(1):1–35, 2005.

[BCG+11] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch Lafuente, and Andrea Vandin. A conceptual framework for adaptation. Submitted, 2011.

[BD07] Meriem Belguidoum and Fabien Dagnat. Dependency management in software component deployment. In Vladimir Mencl and Frank S. de Boer, editors, *FACS*, volume 182 of *Electr. Notes Theor. Comput. Sci.*, pages 17–32, 2007.

[BGM02] Roberto Bruni, Fabio Gadducci, and Ugo Montanari. Normal forms for algebras of connection. *Theor. Comput. Sci.*, 286(2):247–292, 2002.

[BLM06] Roberto Bruni, Ivan Lanese, and Ugo Montanari. A basic algebra of stateless connectors. *Theor. Comput. Sci.*, 366(1-2):98–120, 2006.

[BM02] Roberto Bruni and Ugo Montanari. Dynamic connectors for concurrency. *Theor. Comput. Sci.*, 281(1-2):131–176, 2002.

[BM07] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lect. Notes Comp. Sci.*, pages 18–32. Springer, 2007.

[BMM11a] Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. A connector algebra for p/t nets interactions. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *Lect. Notes Comp. Sci.*, pages 312–326. Springer, 2011.

[BMM11b] Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. Connector algebras, petri nets, and BIP. In Ed Clarke, Irina Virbitskaite, and Andrei Voronkov, editors, *PSI*, volume in press of *Lect. Notes Comp. Sci.* Springer, 2011.

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[BMR01] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based contstraint logic programming: syntax and semantics. *ACM Trans. Program. Lang. Syst.*, 23(1):1–29, 2001.

[BMT10] Maria Grazia Buscemi, Ugo Montanari, and Sonia Taneja. Toward a game-theoretic model of grid systems. In Martin Wirsing, Martin Hofmann, and Axel Rauschmayer, editors, *TGC*, volume 6084 of *Lect. Notes Comp. Sci.*, pages 57–72. Springer, 2010.

[BMT11] Maria Grazia Buscemi, Ugo Montanari, and Sonia Taneja. A game-theoretic analysis of grid job scheduling. Submitted, 2011.

[Bor09] Michele Boreale. Weighted bisimulation in linear algebraic form. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR*, volume 5710 of *Lect. Notes Comp. Sci.*, pages 163–177. Springer, 2009.

[Bru99] Roberto Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems*. PhD thesis, Computer Science Department, University of Pisa, 1999.

[BS08a] Simon Bliudze and Joseph Sifakis. The algebra of connectors - structuring interaction in BIP. *IEEE Trans. Computers*, 57(10):1315–1330, 2008.

[BS08b] Simon Bliudze and Joseph Sifakis. Causal semantics for the algebra of connectors. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *FMCO*, volume 5382 of *Lect. Notes Comp. Sci.*, pages 179–199. Springer, 2008.

[BS10] Simon Bliudze and Joseph Sifakis. Causal semantics for the algebra of connectors. *Formal Methods in System Design*, 36(2):167–194, 2010.

[CDCP11] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party sessions. In Roberto Bruni and Jürgen Dingel, editors, *FMOODS/FORTE*, volume 6722 of *Lect. Notes Comp. Sci.*, pages 1–28. Springer, 2011.

[CGP09] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5), 2009.

[CKM10] Vincenzo Ciancia, Alexander Kurz, and Ugo Montanari. Families of symmetries as efficient models of resource binding. In Bart Jacobs, Milad Niqui, Jan J. M. M. Rutten, and Alexandra Silva, editors, *CMCS*, volume 264(2) of *Electr. Notes Theor. Comput. Sci.*, pages 63–81, 2010.

[CM10] Vincenzo Ciancia and Ugo Montanari. Symmetries, local names and dynamic (de)-allocation of names. *Inf. Comput.*, 208(12):1349–1367, 2010.

[CPCA11] Nicola Capodieci, Andrea Pagani, Giacomo Cabri, and Marco Aiello. Smart meter aware domestic energy trading agents. In *IEEMC*. ACM, 2011.

[CPZ11] Giacomo Cabri, Mariachiara Puviani, and Franco Zambonelli. Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In Waleed W. Smari and Geoffrey C. Fox, editors, *CTS*, pages 508–515. IEEE Computer Society, 2011.

[DLLM09a] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. Marcaspis: a markovian extension of a calculus for services. In Nicola Cannata, Emanuela Merelli, and Irek Ulidowski, editors, *FBTC*, volume 229(4) of *Electr. Notes Theor. Comput. Sci.*, pages 11–26, 2009.

[DLLM09b] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. On a uniform framework for the definition of stochastic process languages. In María Alpuente, Byron Cook, and Christophe Joubert, editors, *FMICS*, volume 5825 of *Lect. Notes Comp. Sci.*, pages 9–25. Springer, 2009.

[DLLM09c] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. Rate-based transition systems for stochastic process calculi. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (2)*, volume 5556 of *Lect. Notes Comp. Sci.*, pages 435–446. Springer, 2009.

[DLLM11] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. State to function labelled transition systems: a uniform framework for defining stochastic process calculi. Technical Report ISTI-2011-TR-012, ISTI, 2011.

[FM97] José Luiz Fiadeiro and T. S. E. Maibaum. Categorical semantics of parallel program design. *Sci. Comput. Program.*, 28(2-3):111–138, 1997.

[FM00] Gian Luigi Ferrari and Ugo Montanari. Tile formats for located and mobile systems. *Inf. Comput.*, 156(1-2):173–235, 2000.

[GM00] Fabio Gadducci and Ugo Montanari. The tile model. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*, pages 133–166. The MIT Press, 2000.

[GMW97] David Garlan, Robert T. Monroe, and David Wile. Acme: an architecture description interchange language. In J. Howard Johnson, editor, *CASCON*, page 7. IBM, 1997.

[Her02] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lect. Notes Comp. Sci.* Springer, 2002.

[HHK02] Holger Hermanns, Ulrich Herzog, and Joost-Pieter Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.

[Hor01] Paul Horn. Autonomic Computing: IBM's perspective on the State of Information Technology, 2001.

[HW11] Matthias Hölzl and Martin Wirsing. Towards a system model for ensembles. In Gul Agha, Olivier Danvy, and José Meseguer, editors, *Festschrift in honor of Carolyn Talcott*, volume 7000 of *LNCS*. Springer, 2011.

[IW95] Paola Inverardi and Alexander L. Wolf. Formal specification and analysis of software architectures using the CHAM model. *IEEE Trans. Softw. Eng.*, 21(4):373–386, 1995.

[KG10] Jung Soo Kim and David Garlan. Analyzing architectural styles. *Journal of Systems and Software*, 83(7):1216–1235, 2010.

[KS86] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1986.

[Mét98] Daniel Le Métayer. Describing software architecture styles using graph grammars. *IEEE Trans. Softw. Eng.*, 24(7):521–533, 1998.

[MK96] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In David Garlan, editor, *SIGSOFT*, volume 21(6) of *ACM SIGSOFT Softw. Eng. Notes*, pages 3–14. ACM, 1996.

[MR99] Ugo Montanari and Francesca Rossi. Graph rewriting, constraint solving and tiles for coordinating distributed systems. *Applied Categorical Structures*, 7(4):333–370, 1999.

[MS11] Ugo Montanari and Matteo Sammartino. Network conscious π-calculus. Submitted, 2011.

[MT75] M. D. Mesarović and Y. Takahara. *General Systems Theory: Mathematical Foundations*, volume 113 of *Mathematics in Science and Engineering*. Academic Press, New York, San Francisco, London, 1975.

[MT97] Nenad Medvidovic and Richard N. Taylor. A framework for classifying and comparing architecture description languages. In Mehdi Jazayeri and Helmut Schauer, editors, *ESEC / SIGSOFT FSE*, volume 1301 of *Lect. Notes Comp. Sci.*, pages 60–76. Springer, 1997.

[MT02] José Meseguer and Carolyn L. Talcott. Semantic models for distributed object reflection. In Boris Magnusson, editor, *ECOOP*, volume 2374 of *Lect. Notes Comp. Sci.*, pages 1–36. Springer, 2002.

[Nas51] John Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[Osb04] Martin J. Osborne. *An introduction to game theory*. Oxford University Press, 2004.

[PW92] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Softw. Eng. Notes*, 17:40–52, 1992.

[RR02] Sriram K. Rajamani and Jakob Rehof. Conformance checking for models of asynchronous message passing software. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV*, volume 2404 of *Lect. Notes Comp. Sci.*, pages 166–179. Springer, 2002.

[Rut07] Jan J. M. M. Rutten. Coalgebraic foundations of linear systems. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *CALCO*, volume 4624 of *Lect. Notes Comp. Sci.*, pages 425–446. Springer, 2007.

[Sob09] Pawel Sobociński. A non-interleaving process calculus for multi-party synchronisation. In Filippo Bonchi, Davide Grohmann, Paola Spoletini, and Emilio Tuosto, editors, *ICE*, volume 12 of *Electr. Proceedings Theor. Comput. Sci.*, pages 87–98, 2009.

[Sob10] Pawel Sobociński. Representations of Petri net interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lect. Notes Comp. Sci.*, pages 554–568. Springer, 2010.

[Ste98] Gheorghe Stefanescu. Reaction and control i. mixing additive and multiplicative network algebras. *Logic Journal of the IGPL*, 6(2):348–369, 1998.

[THK94] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou, and Sergios Theodoridis, editors, *PARLE*, volume 817 of *Lect. Notes Comp. Sci.*, pages 398–413. Springer, 1994.