Using Compositionality to Formally Model and Analyze Systems Built of a High Number of Components

Silvia Bindelli
DEEP-SE Group
DEI, Politecnico di Milano
Milano, Italy
silvia.bindelli@gmail.com

Elisabetta Di Nitto
DEEP-SE Group
DEI, Politecnico di Milano
Milano, Italy
dinitto@elet.polimi.it

Carlo A. Furia
Chair of Software Engineering
ETH Zurich
Zürich, Switzerland
caf@inf.ethz.ch

Matteo Rossi
DEEP-SE Group
DEI, Politecnico di Milano
Milano, Italy
rossi@elet.polimi.it

***Abstract*—When dependability of systems with a large number of components is a concern, being able to model and analyze their properties, especially non-functional ones, in a formal and automated way becomes essential. Often, however, the application of formal methods and automated reasoning is seen by practitioners as complex and time consuming. Compositional techniques can help modify this belief.**

In this paper we show how a compositional modeling and verification technique can be applied to the analysis of distributed systems with numerous interacting nodes. We automate the proof by exploiting a SAT-based tool. We demonstrate the validity of the resulting approach by applying it to an autonomic service-based system that manages, in a coordinated peer-to-peer manner, electricity consumption in a geographical area. In particular, we show that in this case the time needed for performing the proof is remarkably shorter than in the case in which we adopt a non-compositional approach.

***Keywords*-compositionality, formal models and proofs, autonomic distributed systems**

I. Introduction

Highly-distributed systems — such as those that are peer-to-peer, pervasive, or ubiquitous — are typically large systems composed of many elements interacting with one another. When dependability for these systems is a concern, being able to model and analyze their properties, especially non-functional ones, in a way that is as formal and as automated as possible, becomes essential. In these cases, we need to make sure that all possible evolutions of the system do not lead to some invalid or incorrect state.

In the literature various approaches have been proposed to address the issue of modeling and analyzing software-intensive systems. Besides the adoption of general-purpose languages such as those based on logic, on state automata, or on some calculi [1], some authors have suggested the usage of specific languages called Architectural Description Languages (ADLs) [2] and of their associated tools. As discussed in [3], such languages and related tools are not always sufficient to describe and to prove properties of systems that evolve over time in terms of the number of components. In general, formal languages are suitable when the number of components to be considered is relatively small. For large numbers of components, the time needed for the proofs tends to increase significantly and easily becomes too long to be manageable. The problem becomes even more complex when the number of components changes over time.

Compositional techniques can help address these issues.

In this paper, using the compositional framework introduced in [4] as foundation, we present a logic-based technique for the modeling and verification of systems made of a large number of interacting components, such as distributed systems with numerous nodes. The technique is supported by a fully-automated SAT-based tool, which is used to carry out compositional proofs. We demonstrate the validity of the resulting approach by applying it to model and analyze a significantly large embedded and service-based system that manages, in a coordinated peer-to-peer manner, the consumption of electricity in a certain geographical area. In particular, we show that in this case the time needed for performing the proof in the presence of a high number of instances is remarkably shorter than in the case in which we adopt a non-compositional approach.

The paper is structured as follows: Section II introduces the TRIO temporal logic, the ℤot tool that we use for the analysis, and the compositional framework of [4]. Section III presents the modeling guidelines enabling the usage of compositional analysis for distributed systems. Section IV and V describe the case study and its formalization in terms that make it amenable to compositional verification. Section VI presents the compositional verification and compares its performances with those of a non-compositional proof. Section VII discusses related works and Section VIII concludes.

II. Background: TRIO and ZOT

Our compositional modeling and verification approach is based on TRIO [5], a general-purpose formal specification language suitable for describing complex real-time systems, including distributed ones. TRIO is particularly interesting for us as its modularization mechanisms foster the adoption of a compositional modeling approach, which, in turn, is conducive to the development of compositional verification techniques [4]. TRIO is a first-order linear temporal logic

that supports a metric on time. TRIO formulas are built out of the usual first-order connectives, operators, and quantifiers, as well as a single basic modal operator, called Dist, that relates the *current time*, which is left implicit in the formula, to another time instant: given a time-dependent formula F (i.e., a term representing a mapping from the time domain to truth values) and a (arithmetic) term t indicating a time distance (either positive or negative), the formula $\mathrm{Dist}(F, t)$ specifies that F holds at a time instant whose distance is exactly t time units from the current instant. $\mathrm{Dist}(F, t)$ is in turn also a time-dependent formula, as its truth value can be evaluated for any current time instant, so that temporal formulas can be nested as usual. While TRIO can exploit both discrete and dense sets as time domains, in this paper we assume the standard model of the nonnegative integers $\mathbb{N}$ as discrete time domain. For convenience in the writing of specification formulas, TRIO defines a number of *derived* temporal operators from the basic Dist, through propositional composition and first-order logic quantification. Table I defines those used in this paper.

OPERATOR	DEFINITION
$\mathrm{Past}(F, t)$	$t \geq 0 \wedge \mathrm{Dist}(F, -t)$
$\mathrm{Futr}(F, t)$	$t \geq 0 \wedge \mathrm{Dist}(F, t)$
$\mathrm{SomF}(F)$	$\exists d \geq 0 : \mathrm{Futr}(F, d)$
$\mathrm{Alw}(F)$	$\forall d : \mathrm{Dist}(F, d)$
$\mathrm{WithinF}(F, t)$	$\exists d \in [0, t] : \mathrm{Futr}(F, d)$
$\mathrm{Since}(F, G)$	$\exists d > 0 : \mathrm{Past}(G, d) \wedge \forall t \in (0, d) : \mathrm{Past}(F, t)$
$\mathrm{NowOn}(F)$	$\mathrm{Futr}(F, 1)$

Table I
TRIO DERIVED TEMPORAL OPERATORS

The TRIO specification of a system consists of a set of basic *items*, which are primitive elements, such as predicates, time-dependent values, and functions, representing the elementary phenomena of the system. The behavior of a system over time is formally described by a set of TRIO formulas, which state how the items are constrained and how they vary, in a purely descriptive (or declarative) fashion.

To specify large and complex systems, and to support encapsulation, reuse and information hiding at the specification level, TRIO has the usual modular constructs such as classes and modules. The basic encapsulation unit is the *class*, which is a collection of parameters, basic items, and formulas. An instance of a class is, in logical terms, a model of the class formulas (i.e., an interpretation satisfying the conjunction of all formulas). More precisely, every TRIO class C has a set of formulas $\mathcal{F}_C = \{\phi_{C,k}\}_k$ that predicate on its items. A system S in TRIO is described through a set of classes $\{C_i\}_i$. In fact, S is itself a TRIO class, which contains its own formulas $\mathcal{F}_S = \{\phi_{S,j}\}_j$, in addition to those of its components. Then, a system in TRIO is described by the conjunction Σ of the formulas of S and of all the classes it encloses, that is, $\Sigma = \bigwedge_j \phi_{S,j} \wedge \bigwedge_{i,k} \phi_{C_i,k}$ (see [5] for more details on the object-oriented features of TRIO). Section V shows an example of TRIO usage in building a formal model of the energy-sharing example discussed informally in Section IV.

The goal of the verification phase is to ensure that the system S satisfies some desired property R, that is, that $S \models R$. In the TRIO approach S and R are both expressed as logic formulas Σ and ρ, respectively; then, showing that $S \models R$ amounts to proving that $\Sigma \Rightarrow \rho$ is valid.

TRIO is supported by a variety of verification techniques implemented in prototype tools. In this paper we use ℤot [6], a SAT-based bounded satisfiability checker which supports verification of discrete-time TRIO models. ℤot encodes satisfiability (and validity) problems for discrete-time TRIO formulas as propositional satisfiability (SAT) problems, which are then checked with off-the-shelf SAT solvers. Through ℤot one can verify whether stated properties hold for the system being analyzed (or parts thereof) or not; if a property does not hold, ℤot produces a counterexample that violates it.

A. Compositional Verification with TRIO

When dealing with a complex problem, such as the development of a large system, a natural approach to attack it consists in breaking it up in smaller parts, tackle them separately, and then piecing them together to attain the desired result. Such an approach is often called *compositional*, as the goal is achieved by composing intermediate results.

The goal of the compositional approach to verification in TRIO is to simplify the proof $\Sigma \Rightarrow \rho$ by breaking it up into smaller lemmas to be verified for single components C_i, instead of the whole system S. The technical framework to carry out compositional proofs in TRIO has been introduced in [4]. The compositional approach in TRIO is in the so-called *assume/guarantee* mold (see [7, Chap. 4]): the properties ρ_i of a component described through a TRIO class C_i in a system S are proved from the formulas $\mathcal{F}_{C_i}$ of C_i, assuming other properties, $\mathcal{A}_{C_i}$, formalized through *assumption formulas*. Before being able to conclude that ρ_i holds in the whole system S, assumption formulas must be *discharged* (i.e., proved) through the formulas of S (which include those of other components). *Circularities* can occur when, to prove a property ρ_i, a property ρ_j is used whose validity, however, depends on ρ_i. Circularities can be the source of inconsistencies; for this reason, [4] defined an inference rule to soundly prove properties of systems in which certain kinds of circularities arise. In addition, it introduced a sequence of steps through which compositional reasoning can be carried out on TRIO specifications, both in presence and in absence of circularities among properties.

The compositional framework originally defined in [4] was implemented in the PVS theorem prover. This allowed us to check the correctness of the compositional proofs, but the process of proving the properties required a fair amount of interaction with the tool.

III. Using Compositionality for Systems with Many Components

Systems with a modular structure are prime candidates to be modeled and analyzed using compositional techniques. Distributed systems, which are typically made of several components interacting with each other, are intrinsically modular. We focus in particular on peer-to-peer, pervasive and ubiquitous systems as they usually feature a large number of components of a limited number of types. In these kinds of systems (albeit not only in them), components are usually developed with little or no assumptions on the exact physical structure of the overall system, and their interaction with the rest of the system is entirely confined to their interfaces. In this section, using the compositional framework outlined in Section II-A as the theoretical foundation, we describe a formal approach to the modeling and verification of systems with the characteristics mentioned above: a large number of components, few types, and whose interaction with the outside world is encapsulated within their interfaces. The approach is supported by ℤot in a fully automated way. As Section VI shows, the compositional nature of the approach allows us to mitigate the drawbacks of the high computational complexity typical of fully automated verification techniques, and to analyze properties of systems with hundreds of components using a reasonable amount of computational resources.

In our approach, every component c of the system is described by a set of *local properties* $\mathcal{L}_c = \{\lambda_{c,k}\}_k$ (i.e., properties concerning phenomena of the component c). The complete description of a system S is given by all properties of its components, plus the set of its own *global properties* $\mathcal{G} = \{\gamma_j\}_j$. We express properties through sets of suitable formulas $\{\phi_{C,k}\}_k$ and $\{\phi_{S,j}\}_j$ of corresponding TRIO classes. With a slight abuse of notation, let us use $\{\lambda_{c,k}\}_k$ and $\{\gamma_j\}_j$ instead of the more precise $\{\phi_{C,k}\}_k$ and $\{\phi_{S,j}\}_j$. Thus, the complete formal model of system S is simply given by the logical conjunction $\Sigma = \bigwedge_j \gamma_j \wedge \bigwedge_{c,k} \lambda_{c,k}$.

We identify several kinds (i.e., sets) of local properties in a component:

- *internal* properties $\mathcal{I}_c$, which describe features of the component hidden outside of the component itself;
- *shared* properties $\mathcal{S}_c$, which describe features of the component related to its interaction with the outside world;
- *assumptions* $\mathcal{A}_c$, which describe conditions that must hold on the interaction between the component and the outside world for the component to operate correctly.

Then, $\mathcal{L}_c = \mathcal{I}_c \cup \mathcal{S}_c \cup \mathcal{A}_c$. Shared properties and assumptions differ from a methodological point of view in that shared properties must be enforced by the component itself (i.e., they hold if the component behaves properly), while assumptions are enforced by the rest of the system (i.e., they hold if the rest of the system behaves properly). Shared properties and assumptions are part of the *interface* of a component, as their role is to define, from the component's point of view (i.e., locally), its interaction with the rest of the system. Global properties typically describe the actual structure of the system (e.g., the number and types of components it includes), coordination mechanisms among the various components (e.g., sending and delivering of messages between elements of a distributed system), and any other global constraint and domain assumption needed to fully characterize the system. Section V shows formalizations of different kinds of properties for the system studied in this paper.

As mentioned in Section II, the goal of system verification is to show that $\Sigma \Rightarrow \rho$ holds (in logical terms, that it is *valid*), where ρ is the putative property that one wishes to check. In TRIO terms, this means proving that the following TRIO formula is valid.[1]

$$\mathrm{Alw}(\mathcal{G}) \wedge \bigwedge_c \mathrm{Alw}(\mathcal{I}_c \wedge \mathcal{S}_c) \;\Rightarrow\; \mathrm{Alw}(\rho) \tag{1}$$

This check can be performed automatically by the ℤot tool described in Section II. This straightforward process is dubbed "non-compositional verification" in the remainder. However, Formula 1 is quite large, and automated decision procedures have an inherently intractable computational complexity. In fact, the experiments reported in Section VI show that non-compositional verification is practically feasible only for systems with few components.

The compositional approach to verification tackles these limits by breaking Formula 1 into simpler ones such that: (a) successfully checking the simpler formulas guarantees that (1) is valid; (b) the overall effort to tackle the simpler formulas is smaller than the effort to check (1) at once. Local assumptions $\mathcal{A}_s$ are used to enhance the de-coupling between the various modules. Then, compositional verification reduces to the steps in the following inference rule.

1) *Proof of local properties*: check that every module c satisfies locally some suitably-chosen property ρ_c under suitable local assumptions, i.e., check the validity of $\mathrm{Alw}(\mathcal{I}_c \wedge \mathcal{S}_c \wedge \mathcal{A}_c) \Rightarrow \mathrm{Alw}(\rho_c)$. If components belong to a limited number of types, these proofs can be carried out just once for each type of component, thus saving verification effort.
2) *Discharging phase*: check that the assumptions local to every module are satisfied at system level, i.e., check the validity of $\mathrm{Alw}(\mathcal{G}) \Rightarrow \mathrm{Alw}(\mathcal{A}_c)$ for all c.
3) *Proof of global properties*: check that the conjunction of the local properties subsumes global property ρ, i.e., check the validity of $\bigwedge_c \mathrm{Alw}(\rho_c) \Rightarrow \mathrm{Alw}(\rho)$.

One can prove that the above steps are equivalent to checking (1), hence compositional verification is sound (and trivially complete, see [4], [8]).

[1] Assumptions $\mathcal{A}_s$ are implied by $\mathcal{G}$ and the internal and shared properties $\mathcal{S}_{c'}$, $\mathcal{I}_{c'}$ of each component c', if the model is correct.

Notice that the compositional verification approach we envisage can be natually supported by the possibility offered by logics of composing models through conjunction of formulas. Suppose, for example, the system model Σ is made of two parts Γ and Δ, i.e., that $\Sigma = \Gamma \wedge \Delta$. If one can show (e.g., through the ℤot tool) that $\Gamma \Rightarrow \rho$, then it can be trivially concluded that also $\Sigma \Rightarrow \rho$. However, if Γ and Δ are formulas of comparably large size, checking whether $\Gamma \Rightarrow \rho$ is valid is computationally lighter than checking whether $\Sigma \Rightarrow \rho$ is valid.

The 3-step compositional inference rule described above is *non-circular*, as there are no circular dependencies among formulas in the proofs of steps 1–3. In our (and other researchers' [9]) experience non-circular rules, even if less challenging from the theoretical perspective when compared to circular rules, can in practice permit a very efficient and effective verification process.

Depending on the amount of verification carried out at the three steps, results obtained through verification are more or less robust to changes made to the model (which might reflect changes in the actual system, or different configurations to be analyzed). In fact, if the vast majority of proofs is done in step 1 and a component model changes, the re-verification effort is small, as only the part of step 1 concerning the portion of the system that has changed has to be re-done, and steps 2 and 3 are light. If, instead, proofs are mainly carried out at step 3, a change in a component model entails that a significant re-verification effort must be carried out, as the savings in step 1 are small. Ideally, since the classes of distributed systems targeted in this paper are often characterized by a high dynamism, for example due to the fact that components can frequently join/leave the system, we would like the verification process to be at component-level as much as possible, so that the analysis is the least affected by changes in the model. The reader should be warned that obtaining a suitable decomposition for a model is an ingenuity-intensive task that can be burdensome. However, when this is achieved, the verification process can speed up considerably, thus allowing users to use verification as an important tool to guide the design of the system.

Let us stress that we do not advocate compositionality as a "silver bullet" to solve the verification problem for large systems. In fact, while a compositional approach *can* be beneficial in this sense, sometimes, as argued in [10] and [4], the proofs carried out in the three-step process above are not significantly simpler than showing that $\Sigma \Rightarrow \rho$. This can occur for example when the discharging phase is complex enough that it balances any gains obtained by conducting part of the verification process at the component level. Another reason might be that the number of proved local properties ρ_c is comparable to the number of properties in $\mathcal{L}_c$. Also, when the system is made of many different types of components with few instances each, the number of local properties ρ_c to be proved increases, thus reducing the benefits of employing a compositional approach.

IV. The energy-sharing case-study

The energy-sharing case study targets energy sharing among residential homes. It relies on the observation that while energy consumption in a building is not constant all the time, the way the electrical power distribution system is organized makes any change quite difficult to handle. In general, from the point of view of the energy provider, unforeseen peaks are difficult to manage, unless the production as well as the distribution system to each consumption unit is designed by overestimating the consumption. In Italy, the issue is currently addressed by having meters installed at each site, which check for power consumption and stop the provision of electricity if the consumption goes above a certain threshold. In the line of the principle that sharing resources within social communities helps sustainability, we could envisage a more dynamic solution that is able to internally auto-adapt, to some extent, to the variations of power consumption at the various individual sites that belong to the community. This helps in guaranteeing that the overall consumption remains within some established limits.

We have developed a community of sites each controlled by some software component in charge of monitoring the power consumption within a site and of requesting and offering help to the other sites when needed. Each site can consume more than its threshold provided that, at the same time, there exist some other sites consuming less and available to lend part of their threshold. The overall goal of optimizing the energy consumption is not achieved through a centrally-supervised execution, but it emerges at system-level from the collaboration among various components.

The system is described in [11] and is based on an autonomic infrastructure and architectural model called `SelfLet`s. According to the `SelfLet`s model a software system is composed of various autonomous elements, each of which is a `SelfLet`. Every `SelfLet` is defined in terms of: 1) a *main behavior* that is continuously executed by the component, 2) a number of other behaviors that can be executed in parallel with the main behavior and are triggered on demand by the component itself or by its neighbors through a service call, and 3) an *autonomic policy* that defines some reactions to abnormal situations that can occur during the life-time of the `SelfLet`. The reactions include the possibility of changing the main behavior, disabling/enabling the execution of the other behaviors, disabling/enabling the interaction with some neighbors, etc. Designers program the behaviors using UML state diagrams (with some restrictions that have been introduced to formalize their semantics) and the autonomic policy using the Drools [12] rule language.

For the sake of brevity we do not provide many details on the `SelfLet`s approach, but we briefly describe the behavior of the components of the electricity-sharing example, which are modeled and analyzed formally in the rest of this

paper. Figure 1 shows the main behavior of an *Electricity-Sharing* `SelfLet`. State `needSenseConsumption` indicates that the `SelfLet` needs to sense the current energy consumption; for brevity we do not discuss how this is achieved in the implementation.

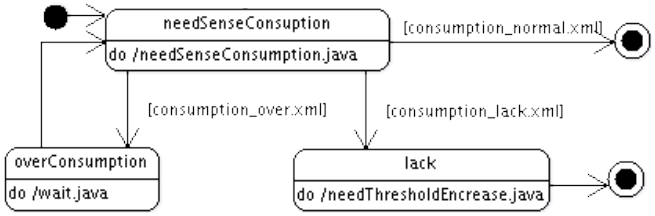

Figure 1. Main behavior of the Electricity `SelfLet`.

Once data about consumption have been gathered, three possible scenarios can take place: the consumption is safely below the threshold (`consumption_normal`), the consumption is dangerously close to the threshold (`consumption_lack`) or it is greater than the threshold (`consumption_over`). Both a normal consumption and an over consumption eventually restart the `SelfLet` in the `needSenseConsumption` state, possibly after an handling an exception in the case of over consumption.

The most interesting state is `lack`. In this state a request for the service named `thresholdIncrease` is issued. Such a service can be offered by some other `SelfLet` whose current energy consumption is well below the threshold. We have implemented a specific autonomic policy that sets the `thresholdIncrease` service as executable by the `SelfLet` when the consumption is low, and disables it when the consumption is high. This policy implements the idea that a `SelfLet` can "lend" part of its threshold only when it does not need all of it. The policy is composed of one rule called "changed consumption reaction" that is triggered when the `consumption` property is updated. The behavior implementing service `thresholdIncrease` offers part of its threshold to the requester according to the formula $output = ((threshold - consumption)/2)$; then, the offering `SelfLet` can lend only part of its unused threshold, limiting the possibility that it will soon enter a `lack` state.

From the deployment point of view, the system is composed of an unlimited number of Electricity-Sharing `SelfLet` replicas, each of them installed in some location (home, industrial site, ...) that is participating to the energy sharing. The various replicas are connected through some network so that they can interact to exchange their threshold. The model of Section V formalizes the fundamental behavioral aspects of `SelfLet`s sketched in this section.

V. The energy-sharing example in TRIO

The formal `SelfLet` model is developed with a compositional approach by exploiting TRIO's modular features. This amounts essentially to: identifying the fundamental components of the system; describing their internal and shared properties, according to the distinction drawn in Section III; and formalizing how they are composed into a global system. The formal model is centered around a `SelfLet` TRIO class describing the fundamental behavior of a `SelfLet` through TRIO formulas.[2]

A. The `SelfLet` class

According to the distinction introduced in Section III, the formulas of every `SelfLet` s are partitioned into three sets $\mathcal{I}_s, \mathcal{S}_s, \mathcal{A}_s$. They denote respectively *internal*, *shared*, and *assumption* properties. Let us start with set $\mathcal{I}_s$ of formulas which formalize the behavior that is totally independent of communication with the environment.

Internal properties $\mathcal{I}_s$. A first layer of description consists in the formalization of the state diagram of Figure 1 augmented with some additional states and transitions that account for the presence of the `SelfLet` runtime. The state diagram that is actually modeled in TRIO is shown in Figure 2. It differs from the original one only for states `normal` and `wait` and for the transitions that connect these to the other states. State `wait` introduces a delay in the re-execution of the main behavior of Figure 1, while state `normal` indicates a consumption within the normal range.

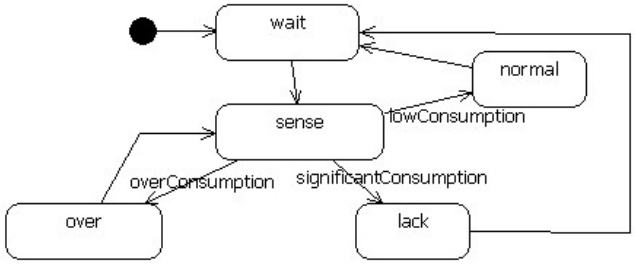

Figure 2. The state machine modeled in TRIO.

According to a standard state-based modeling, at any instant of time the configuration of a `SelfLet` s is characterized by the current state, formalized by time-dependent item sLetState with domain $\{\mathsf{lack}, \mathsf{wait}, \mathsf{over}, \mathsf{normal}, \mathsf{sense}\}$. Transitions between states are triggered by suitable events. For instance, consider state over which can be exited while going to state sense. Time-dependent predicates toSense, toOver represent the events of going to state sense and state over in the next instant. Hence, state over is characterized by:

$$\mathsf{sLetState} = \mathsf{over} \;\Leftrightarrow\; \mathrm{Since}(\neg\mathsf{toSense}, \mathsf{toOver}) \quad (2)$$

Similar formulas describe the other states over which sLetState ranges. In addition to discrete state sLetState, the overall configuration of a `SelfLet` is characterized by the current values of energy consumption and threshold, formalized by items consumption and threshold ranging

[2]The complete model used for verification with the Zot tool can be found at `http://se.inf.ethz.ch/people/furia/misc/sct.tgz`.

over the discrete sets $[\mathrm{MIN_C}, \mathrm{MAX_C}]$ and $[\mathrm{MIN_T}, \mathrm{MAX_T}]$, respectively (with $\mathrm{MIN_C}, \cdots, \mathrm{MAX_T}$ suitable constants). The state-space of consumption and energy values is partitioned into three sets, represented by the mutually-exclusive time-dependent predicates $\mathsf{cCrit}, \mathsf{cOver}, \mathsf{cNormal}$, according the current values of threshold and consumption. Namely, cCrit means that the `SelfLet` is lacking energy, hence consumption is close to threshold:

$$\mathsf{cCrit} \Leftrightarrow \frac{95}{100}\mathsf{threshold} \leq \mathsf{consumption} \leq \mathsf{threshold} \quad (3)$$

Similarly, cOver holds iff $\mathsf{consumption}$ is greater than $\mathsf{threshold}$; and $\mathsf{cNormal}$ holds in the remaining case, i.e., when $\mathsf{consumption}$ is less than 95% of $\mathsf{threshold}$.

Other internal properties, not shown here, specify how the values of $\mathsf{threshold}$ and $\mathsf{consumption}$ are updated over time.

Shared properties $\mathcal{S}_s$. The consumption of a `SelfLet` depends on the electricity demand of its appliance at any given time; we model this conservatively by allowing $\mathsf{consumption}$ to change nonderministically within the range $[\mathrm{MIN_C}, \mathrm{MAX_C}]$. On the contrary, $\mathsf{threshold}$ is a "shared resource" in that it can only be exchanged among `SelfLets`, according to the following shared formulas.

Time-dependent predicates $\mathsf{offerTInc}$ and $\mathsf{needTInc}$ represent when a `SelfLet` signals to other `SelfLets` that it offers or needs to receive some threshold, respectively: whenever the `SelfLet` is functioning normally and its threshold is above the minimum, its threshold is offered; whenever it is lacking threshold, some threshold is needed.

$$\mathsf{offerTInc} \;\Leftrightarrow\; \mathsf{cNormal} \wedge \mathsf{threshold} > \mathrm{MIN_T} \quad (4)$$

$$\mathsf{needTInc} \;\Leftrightarrow\; \mathsf{cCrit} \quad (5)$$

Predicate $\mathsf{offerTInc}$ models the autonomic policy described in Section IV. Time-dependent predicate $\mathsf{exchange}(s, s', \delta)$ signals when a threshold exchange with amount δ occurs between `SelfLet` s and `SelfLet` $s' \neq s$. Time-dependent predicate exchange is not constrained "locally" by formulas in $\mathcal{I}_s$, $\mathcal{S}_s$, or $\mathcal{A}_s$ because it pertains to the global dynamics of the system. However, it constrains the local changes to the threshold, since threshold changes iff it is exchanged:[3]

$$\exists s' : \mathsf{exchange}\big(s, s', \delta\big) \;\Leftrightarrow\; \mathsf{needTInc} \wedge \mathsf{TIncd}(\delta) \quad (6)$$

$$\mathsf{TIncd}(\delta) \wedge \mathsf{threshold} = \tau \Rightarrow \mathrm{NowOn}(\mathsf{threshold} = \tau + \delta) \quad (7)$$

Notice that the exact amount of offered threshold is chosen nondeterministically (within a fixed range Δ). Hence, the properties proved in Section VI also hold for the more constrained policy described in Section IV.

Assumptions $\mathcal{A}_s$. The local assumption states that whenever a `SelfLet` requires a threshold increase, sooner or later some other `SelfLet` will offer to help it:

$$\mathsf{ASS}_s : \quad \mathsf{needTInc} \Rightarrow \mathrm{SomF}(\exists \delta \in \Delta : \mathsf{TIncd}(\delta))$$

[3]Predicate $\mathsf{TIncd}(\delta)$ holds in all those instants in which there is an increase of δ in the threshold.

In other words, the `SelfLet` assumes cooperating neighbors, in number sufficient to guarantee that not all `SelfLets` will be lacking threshold at the same time.

B. The overall system

The overall system is a collection of N_s `SelfLet` modules — i.e., independent instances of the `SelfLet` TRIO class — $\mathsf{SelfLets}[s]$ with $s \in [1..N_s]$. The overall system introduces an additional set of global properties $\mathcal{G}$, which formalize coordination mechanisms among `SelfLets`.

For any two `SelfLets` s_1, s_2, $\mathsf{exchange}(s_1, s_2, \delta)$ implies that: (1) s_1 increases its threshold by δ, and (2) s_2 decreases its threshold by δ.

$$\mathsf{exchange}(s_1, s_2, \delta) \Leftrightarrow \left(\begin{array}{l} \mathsf{SelfLets}[s_1].\mathsf{TIncd}(\delta) \\ \wedge\ \mathsf{SelfLets}[s_2].\mathsf{TDecd}(\delta) \end{array} \right) \quad (8)$$

In addition, if there are two `SelfLets` s_1, s_2 such that s_2 is offering threshold and s_1 is requiring it, a threshold exchange between *some* two `SelfLets` occurs within some T_e time units, with T_e a suitable constant. This abstract synchronization mechanism does not require that the pair of `SelfLets` exchanging the threshold are the same of the premises as long as it is a suitable "need/offer" pair.

$$\left(\begin{array}{c} \exists s_1, s_2 : \\ \mathsf{SelfLets}[s_1].\mathsf{needTInc} \wedge \\ \mathsf{SelfLets}[s_2].\mathsf{offerTInc} \end{array} \right) \Rightarrow \left(\begin{array}{c} \exists s_1', s_2', \delta \in \Delta : \\ \mathsf{SelfLets}[s_1'].\mathsf{needTInc} \wedge \\ \mathsf{SelfLets}[s_2'].\mathsf{offerTInc} \wedge \\ \mathrm{WithinF}\big(\mathsf{exchange}\big(s_1', s_2', \delta\big), T_e\big) \end{array} \right) \quad (9)$$

VI. Applying compositionality: approach and findings

This section presents the properties to be proven and the outcome of their automated verification, comparing in particular the non-compositional and compositional approaches.[4]

While a `SelfLet`'s $\mathsf{threshold}$ and $\mathsf{consumption}$ values are constrained by rules, they interact indirectly in complex ways. Hence, we would like to ensure that threshold values do not grow arbitrarily, as a threshold increment can be performed only when it is required, and it is required only if the threshold is sufficiently low w.r.t. the consumption.

$$\mathsf{P}_1 : \bigwedge_{1 \leq s \leq N_s} \mathsf{SelfLets}[s].\mathsf{threshold} < 2 \cdot \mathsf{SelfLets}[s].\mathrm{MAX_C}$$

The significance of property P_1 is two-fold. First, it establishes that the restriction to the domain of $\mathsf{threshold}$ to a finite range is without loss of generality, as $\mathsf{threshold}$ does not grow indefinitely anyway. Second, it demonstrates that `SelfLets` behave in a fair, "non-greedy" manner, and they do not hoard threshold, and starve other `SelfLets` for it.

Another expectation we have on any system of multiple interacting `SelfLets` is that they sustain one another in such a way that the overall system provides a sufficiently

[4]All tests have been performed on AMD Athlon64 X2 Dual Core Processor 4000+ with 2 Gb of RAM, Kubuntu GNU/Linux (kernel 2.6.22), Zot with SBCL v. 1.0.11 and MiniSat v. 2.0.

high level of service. Property P_2 formalizes a significant aspect of this requirement by demanding that no `SelfLet` stays indefinitely in a lack state.

$$\mathsf{P}_2: \quad \bigwedge_{1 \le s \le N_s} \mathrm{SomF}(\neg \mathsf{SelfLets}[s].\mathsf{cCrit})$$

Somewhat surprisingly, automatic verification through ℤot showed that property P_2, is *not* a valid property of the system. This is the case even if the system is initialized with a sufficient total amount of threshold, so that not all `SelfLet`s can be lacking at the same time. In fact, counter-examples produced by ℤot showed subtle — yet possible — unfair scenarios, where two (or more) `SelfLet`s help each other while completely ignoring a third one, which cannot exit its lacking state because no one gives it any threshold. From the counter-examples it can be concluded that, if the system does not have some kind of fairness property guaranteeing that, if a `SelfLet` needs threshold, it will eventually receive some, property P_2 cannot hold. Since P_2 is a desirable property of our system, we seek suitable further constraints to add to the model, which make P_2 valid. Such constraints must then be reflected on the actual system implementation, so they have to be carefully chosen, lest they are unrealizable in practice. Hence, we avoid constraining *a priori* the maximum number of `SelfLet`s that are lacking at any time, nor we introduce a centralized arbiter to ensure fairness of threshold exchanges, as this would make the whole system no more highly-distributed, and would probably render compositional verification mostly ineffective, as much of the proof burden would be concentrated on the single arbiter, or at system-level. Instead, we introduce global constraint $\mathsf{B} \in \mathcal{G}$: every request for a threshold increase is partially satisfied (by some unspecified `SelfLet`) within $2T_r$ time units (with T_r a suitable constant).

$$\mathsf{B}: \bigwedge_{1 \le s \le N_s} \left(\begin{array}{c} \mathsf{SelfLets}[s].\mathsf{needTInc} \\ \Rightarrow \\ \mathrm{WithinF}(\exists \delta \in \Delta : \mathsf{SelfLets}[s].\mathsf{TIncd}(\delta), 2T_r) \end{array} \right)$$

Notice that this does not make P_2 vacuously true, as exiting a lack state requires more than one threshold increase in the most general case, and multiple threshold exchanges may interleave subtly. B induces implicitly a maximum number of lacking `SelfLet` at any time, but it does not determine how many. In addition, it is nicely compositional in that it allows a very effective and clean decoupling of modules (in the sense used in Section III). Below we verify that P_2 holds for any system of `SelfLet`s that also satisfies B. We will then identify a weaker constraint that still makes P_2 true.

As mentioned in Section III, non-compositional verification amounts to using ℤot to check the validity of (1) with P_1 and P_2 substituting ρ. The results of this process are summarized in Table II. For each test the table reports: the checked property (P_1 or P_2); the number N_s of `SelfLet`s in the system; the size k of time bounds (as ℤot is a bounded satisfiability checker); the total amount of time (in minutes) and space (in Mb) needed for executing the proof.

PROPERTY	N_s	k	TIME	SPACE
P_1	5	10	47.74	566.6
P_1	5	20	180.29	616.3
P_1	5	30	512.50	948.7
P_2	3	10	5.09	186.1
P_2	3	20	21.76	368.5
P_2	3	30	47.45	551.7
P_2	5	10	48.46	568.9
P_2	5	20	192.03	616.4
P_2	5	30	436.34	917.7

Table II
NON-COMPOSITIONAL VERIFICATION.

The results show clearly that non-compositional verification has serious scalability limitations. In practice, the verification of systems with more than 5 interacting `SelfLet`s is already extremely resource-consuming. This is no surprise, as exhaustive verification techniques suffer from the well-known state-explosion problem, hence more sophisticated and *ad hoc* approaches are required to achieve scalability. Compositionality provides precisely such scalability.

Let us now apply the generic compositional proof steps of Section III to carry out the verification of P_1 and P_2.

The proof of P_1 can be carried out entirely locally to the `SelfLet` modules, without the need of additional assumptions. Intuitively, any `SelfLet` stops asking for threshold increases if its threshold is sufficiently high. This behavior is guaranteed regardless of what other `SelfLet`s do, hence it can be proved at component-level. More technically, the verification of P_1 reduces to checking the validity of:

$$\mathrm{Alw}(\mathcal{I}_s) \;\Rightarrow\; \mathrm{Alw}(\mathsf{threshold} < 2 \cdot \mathrm{MAX_C}) \qquad (10)$$

for all $s \in [1..N_s]$. This corresponds to step 1 of Section III with ρ_s being $\mathsf{threshold} < 2 \cdot \mathrm{MAX_C}$ for all s. Step 2 is empty because we use no assumptions, and step 3 is trivial because it is easy to see that $\bigwedge_s \rho_s \equiv \mathsf{P}_1$. Given that all `SelfLet` modules are identical, the whole proof amounts to a single check of (10) for an arbitrary `SelfLet` $s \in [1..N_s]$. Hence the verification effort does not depend on the number of `SelfLet`s N_s in the system, a dramatic improvement over the non-compositional approach.[5]

The results of this compositional verification of property P_1 are summarized in Table III; the third column reports the SCALE factor by which all temporal constants in the systems are multiplied. Given that the required resources do not depend on the number of `SelfLet`s N_s, it is possible to analyze systems with a virtually unlimited number of `SelfLet`s with very reasonable resources; also notice that there is no time/space trade-off with respect to the non-compositional case, which is far worse according to both

[5]Even if every `SelfLet` were different than the others, the verification time would still scale *linearly* with N_s, as verification would simply entail repeating step 1 N_s times, since step 2 is empty.

measures. Further experiments vary the SCALE and bound k parameters, which determine the size of formulas for single SelfLets. While the required verification resources change, as expected, even the largest attempted tests terminated within reasonable time.

N_s	k	SCALE	TIME	SPACE
3	10	1	0.35	46.0
3	30	3	2.68	137.1
3	100	10	36.76	504.3
10	10	1	0.36	46.0
10	20	1	1.20	90.3
10	30	5	2.85	144.2
50	10	1	0.35	46.0
50	20	1	1.22	90.3
50	30	2	2.68	135.8
100	20	1	1.20	90.3
100	30	1	2.60	134.0
100	50	3	7.35	227.7
100	80	1	18.47	349.9
100	80	3	19.91	377.4
100	100	1	29.13	445.3
100	100	3	30.64	454.6
500	20	1	1.20	90.3

Table III
COMPOSITIONAL VERIFICATION OF P_1.

The application of compositionality to the verification of P_2 exploits the local assumption defined in Section V. Essentially, the system-wide property P_2 is split into a set of single assumptions, each local to each SelfLet; this decouples the various SelfLets so that the actual verification can be done almost entirely at the local level. Then, the three compositional proof steps of Section III are instantiated into the following three formulas (for all $s \in [1..N_s]$).

$$\mathrm{Alw}(\mathcal{I}_s \wedge \mathcal{S}_s \wedge \mathcal{A}_s) \Rightarrow \mathrm{SomF}(\neg \mathsf{cCrit}) \quad (11)$$

$$\mathrm{Alw}(\mathcal{G}) \Rightarrow \mathrm{Alw}(\mathsf{ASS}_s) \quad (12)$$

$$\bigwedge_{1 \le s \le N_s} \mathrm{Alw}(\mathrm{SomF}(\neg \mathsf{SelfLets}[s].\mathsf{cCrit})) \Rightarrow \mathrm{Alw}(\mathsf{P}_2) \quad (13)$$

Similarly as for P_1, (13) is trivial because it is easy to see that $\bigwedge_s \rho_s \equiv \mathsf{P}_2$, where ρ_s is $\mathrm{SomF}(\neg \mathsf{cCrit})$ for any s in this case. (12) is typically simple to establish, because $\mathsf{B} \in \mathcal{G}$ alone subsumes any ASS_s. (11) is instead more intricate, but it can be checked independently of the number N_s of SelfLets because all SelfLets are identical. Table IV summarizes the results of this compositional verification of property P_2 for different values of N_s, k, and scale (SC) factor. For every test, the time and space (T/S) of both steps 1 and 2 (corresponding to (11) and (12), respectively) is reported, as well as the totals for the whole compositional verification of P_2. This shows that the complexity of the component-level check of (11) is independent of N_s, as for P_1. On the other hand, the complexity of checking (12) grows (also) with N_s. As a result, whereas the relative weight of checking (12) over the whole verification process is negligible for small values of N_s, it becomes dominant roughly for $N_s > 50$. The other parameters k and SC influence, instead, both verification steps, as they affect the size not only of the whole interacting system but also of single SelfLets. In all, the compositional approach allowed us to verify P_2 for systems of significantly large size with resources similar to those required for only 5 SelfLets and a bound of 30 in the non-compositional tests.

N_s	k	SC	T/S (11)	T/S (12)	TOT. T/S
3	10	1	0.37 / 49.3	0.02 / 6.1	0.39 / 49.3
3	30	3	2.88 / 139.2	0.08 / 20.9	2.96 / 139.2
3	100	10	39.06 / 516.3	2.92 / 150.1	41.98 / 516.3
10	10	1	0.36 / 49.3	0.05 / 14.3	0.41 / 49.3
10	20	1	1.30 / 91.3	0.10 / 26.0	1.40 / 91.3
10	30	5	3.09 / 146.7	0.88 / 85.8	3.97 / 146.7
50	10	1	0.36 / 49.3	0.48 / 61.7	0.84 / 61.7
50	20	1	1.30 / 91.3	1.50 / 113.6	2.81 / 113.6
50	30	2	2.79 / 137.2	5.94 / 234.8	8.73 / 234.8
100	20	1	1.33 / 91.3	5.88 / 231.5	7.21 / 231.5
100	30	1	3.01 / 135.5	12.42 / 344.8	15.43 / 344.8
100	50	3	7.76 / 230.5	144.13 / 473.0	151.89 / 473.0
100	80	1	19.24 / 353.0	152.64 / 452.3	171.88 / 452.3
100	80	3	20.77 / 382.0	375.60 / 787.4	396.37 / 787.4
100	100	1	31.21 / 448.5	224.85 / 561.5	256.06 / 561.5
100	100	3	31.83 / 461.2	582.38 / 930.5	614.21 / 930.5
500	20	1	1.43 / 91.3	583.17 / 571.7	584.59 / 571.7

Table IV
COMPOSITIONAL VERIFICATION OF P_2.

We can at this point study P_2 and B more deeply. In particular, one can show that constraint B is strong enough to guarantee that a SelfLet cannot stay in state cCrit for more than $10T_r$ time units. The new property to be proved becomes, in this case, $\forall(1 \le s \le N_s) : \mathrm{WithinF}(\neg \mathsf{SelfLets}[s].\mathsf{cCrit}, 10T_r)$ and Zot checks its validity in 26 minutes for 100 SelfLets, with $k = 30$. This check suggests that B is likely stronger than needed to prove the original P_2 property. In fact, if we relax constraint B by replacing the WithinF operator with SomF (which entails that the delay with which a SelfLet in need of threshold receives it is not bounded) we show that property P_2 still holds (Zot checks its validity in 19 minutes for 100 SelfLets, with $k = 30$). The new constraint that we obtain can be implemented in the actual system more easily than the original one. In fact, it is possible to introduce in each SelfLet a mechanism that, when this SelfLet is in need of threshold, gradually increases its priority and that, when it is offering part of the threshold, tends to privilege the requests from SelfLets with higher priority.

Once the effectiveness of the compositional verification approach is established, the model can be used to explore new properties, or to try different system parameters.

An interesting analysis concerns the accuracy of the discretization introduced in the formal model. Items consumption and threshold model physical quantities, hence they should be ideally real-valued items (and the range of threshold would even be unbounded). However, the formal model must approximate these real quantities by a finite set of values in order to make automated finite-state verification possible. In the experiments discussed so far we have introduced a rather coarse finitization of the ranges, where consumption and threshold vary over the finite sets

of integers $[0..5]$ and $[1..11]$, respectively. This is essentially the coarsest finitization that does not render the model trivial (and where in particular SelfLets require more than one threshold increase to exit the critical range). Therefore, it may be interesting to extend these ranges in order to make the formal model less abstract and the finitization finer.

k	consumption	T/S (11)	T/S (12)	TOT. T/S
50	[0..8]	12.93/246.8	0.07/14.2	12.99/260.9
50	[0..10]	22.73/316.0	0.06/14.2	22.79/330.2
50	[0..12]	37.76/398.4	0.07/14.2	37.83/412.5
50	[0..15]	173.59/325.2	0.06/14.2	173.65/339.3
70	[0..8]	27.12/329.1	0.11/19.5	27.23/348.6
70	[0..10]	43.75/442.1	0.10/19.5	43.85/461.5
70	[0..12]	168.88/324.9	0.11/19.5	168.99/344.4
70	[0..15]	388.03/437.6	0.10/19.5	388.14/457.1
100	[0..8]	109.33/266.4	0.16/26.5	109.49/292.9
100	[0..10]	238.39/375.6	0.17/26.5	238.56/402.1

Table V
COMPOSITIONAL VERIFICATION OF P_2 FOR 3 SELFLETS.

We ran one more set of experiments where we varied the ranges of consumption and threshold in the compositional verification of P_2; the results are reported in Table V. In all these experiments we considered a system of just 3 SelfLets, and the range of item threshold is set to $[1..2\mathrm{MAX_C}]$, where $\mathrm{MAX_C}$ is the maximum value for item consumption. Since we limit the number of SelfLets in the system, resources needed to prove step (12) in the compositional proofs are negligible, whereas all effort is for step (11) which instead depends on the ranges of the items. The experiments show clearly that the detailed model is more expensive than the original one in terms of the verification effort. In particular, non-compositional proofs are unfeasible even with a mere 3 SelfLets, and one can scale up to larger ranges only by applying the compositional technique in the first place. These tests provide even more evidence of the importance of compositionality in system analysis, and demonstrate that one can verify non-trivial system models only if the modularization of the system is exploited in the verification process as accurately as possible.

A. Findings and threats to validity

The experiment we have presented in this section has allowed us to identify a subtle problem of its implementation. The actual system, in fact, has been built assuming that property P_2 would have held without introducing any specific fairness mechanism. As we have highlighted before, the first result we have obtained with the analysis has been the identification of this lack of fairness. The execution of analyses with various revisions of property P_2 and of the constraint B has allowed us to start reasoning on the issue also from the perspective of the actual implementation of the solution. Thanks to the fact that the analysis has been performed with a fully-automated tool, we have been able to easily and quickly run the model in various situations and explore it in depth. Had we adopted the PVS approach used in [4] such exploration would have required a significant manual effort and specific analysis skills. The introduction of compositionality not only for the modularization of the model, but, mainly, for its analysis, has allowed us to prove the properties in the presence of a large number of components. Moreover, we have shown that through compositionality it is also possible to explore large domains for variables, provided that the number of components in this case is kept small.

Of course, some threats to validity need to be investigated as well. First of all, even if we refer to a system with a large number of SelfLets, we are only considering a single type of SelfLet. The results we obtain, however, can be still considered valid even for a higher number of types, if this is kept an order of magnitude smaller than the number of instances and if assumptions $\mathcal{A}_s$ are well designed. As discussed in Section III, the local proofs will have to be executed for each type, and therefore, their number will increase linearly with the number of considered types. Another potential threat concerns the limitation in the number of SelfLets we had to introduce when increasing the domains of variables. This is due to the inherent complexity of SAT-based proofs and model checking techniques. However, the "push-button" approach offered by such techniques allows us to govern the complexity by varying one parameter at a time when needed.

VII. RELATED WORK

In the literature, a wide array of compositional inference rules, which allow one to conclude the validity of a property of a system from the validity of properties of its components, have been devised. The interested reader can refer to [7, Chap. 4] for a survey of some relevant approaches. Despite the abundance of compositional inference rules, however, their application to the analysis of large systems remains spotty. In this paper, with the aid of an automated SAT-based verification tool, we have been able to prove some meaningful properties of a large pervasive system.

In recent years, techniques have been developed to automatically infer assumptions for components modeled through finite state machines (or transition systems) [13]. They have been successfully applied to the automated compositional analysis of autonomous systems [14], though, in their current state, their practical effectiveness and improvements with respect to monolithic verification has been questioned in [15]. The analysis technique applied in this paper is automated in the sense that property verification is carried out through a SAT-based, fully automated, tool, whereas assumptions have been produced manually. This implies that, currently, the technique used here relies heavily on the modeler's expertise; however, as Section VI showed, the improvements obtained over noncompositional verification are significant. In fact, we were able to check the validity of the desired properties for systems with orders of magnitude more components than those that were analyzable through

monolithic verification, while [15] showed that techniques based on automatic inference of assumptions are often able to manage only few (if any) more components than noncompositional verification.

In [16] compositionality is used to guarantee that, in a middleware-based distributed system, the abstract properties of the system (which is modeled through transition systems composed together) that hold when the middleware is not considered still hold after the latter is introduced in the model. Along similar lines, [17] uses model checking to verify properties of a complex system while considering the constraints of the resources used for its execution. Hence, the results in [16] and [17] are in a way complementary to those presented in this paper. In fact, we focus on proving properties of the system using a very abstract model of the communication mechanism between components.

VIII. Conclusion

We have discussed in this paper on how to apply a compositional technique to the proof of properties of systems with a high number of components. The results we have obtained referring to the energy-sharing example show that the approach is remarkably scalable and able to provide results in a significantly small amount of time even when the proof involves a high number of components. We have also shown the advantage of the push-button proof style offered by Zot. It is surely more user-friendly, compared to a proof checker, and allows system designers to simplify and speed up the verification process in many cases.

The energy-sharing example, even though simplified in this paper, presents interesting characteristics that are common also to other systems. In particular, it is composed of a high number (not fixed *a priori*) of distributed components that interact with a peer-to-peer logic without any centralization point. Moreover, the behavior of each peer evolves over time depending on actual consumption of energy at the local site and on the requests coming from the other sites. Thus, we argue that the specific results we have obtained in this case can be obtained in other cases, for systems with these distinguishing features. Clearly, as the reader can guess, if peers share the same characteristics (i.e., are of the same type) step 1 of the compositional proof is executed only once to prove the local properties of all components. Viceversa, if peers are completely different from each other in terms of their main characteristics, the step 1 of the compositional proof will have to be repeated several times to prove the (different) properties of all peers. This would increase the cost of the proof in terms of time, but, if the system is well modularized, it could still be possible to obtain an advantage from the relatively fast execution of steps 2 and 3.

As future work we plan to consolidate the case study by enriching its model and verifying new properties. Our long term objective is to build a general design and verification environment for the `SelfLets` autonomic framework.

Acknowledgements: This work has been partially supported by the PRIN D-ASAP (2007XKEHFA) and the IDEAS-ERC SMScom (227977) projects. We thank Judith Bishop and the reviewers for their valuable comments.

References

[1] C. L. Heitmeyer and D. Mandrioli, Eds., *Formal Methods for Real-Time Computing*. J. Wiley & Sons, 1996.

[2] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE TSE*, vol. 26, no. 1, pp. 70–93, 2000.

[3] E. Di Nitto and D. Rosenblum, "Exploiting ADLs to specify architectural styles induced by middleware infrastructures," in *Proc. ICSE*, 1999, pp. 13–22.

[4] C. A. Furia, M. Rossi, D. Mandrioli, and A. Morzenti, "Automated compositional proofs for real-time systems," *Th. Comp. Sci.*, vol. 376, no. 3, pp. 164–184, 2007.

[5] E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti, "From formal models to formally-based methods: an industrial experience," *ACM TOSEM*, vol. 8, no. 1, pp. 79–113, 1999.

[6] M. Pradella, A. Morzenti, and P. S. Pietro, "The symmetry of the past and of the future," in *Proc. ESEC/FSE*, 2007, pp. 312–320.

[7] C. A. Furia, "Scaling up the formal analysis of real-time systems," Ph.D. dissertation, Politecnico di Milano, 2007.

[8] ——, "A compositional world: a survey of recent works on compositionality in formal methods," DEI, Politecnico di Milano, Tech. Rep. 2005.22, 2005.

[9] C. S. Păsăreanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer, "Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning," *Form. Meth. in Sys. Des.*, vol. 32, no. 3, pp. 175–205, 2008.

[10] L. Lamport, "Composition: A way to make proofs harder," in *Compositionality: The Significant Difference*, ser. LNCS, vol. 1536, 1998, pp. 402–423.

[11] S. Bindelli, E. D. Nitto, R. Mirandola, and R. Tedesco, "Building autonomic components: the SelfLets approach," in *Proc. ASE - Workshops*, 2008, pp. 17–24.

[12] `http://www.jboss.org/drools/`.

[13] D. Giannakopoulou and C. S. Păsăreanu, "Special issue on learning techniques for compositional reasoning," *Form. Meth. in Sys. Des.*, vol. 32, no. 3, pp. 173–174, 2008.

[14] G. Brat, E. Denney, D. Giannakopoulou, J. Frank, and A. Jonsson, "Verification of autonomous systems for space applications," in *Proc. of the IEEE Aerospace Conf.*, 2006.

[15] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke, "Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning," *ACM TOSEM*, vol. 17, no. 2, pp. 7:1–7:52, 2008.

[16] M. Caporuscio, P. Inverardi, and P. Pelliccione, "Compositional verification of middleware-based software architecture descriptions," in *Proc. ICSE*, 2004, pp. 221–230.

[17] H. Foster, W. Emmerich, J. Kramer, J. Magee, D. Rosenblum, and S. Uchitel, "Model checking service compositions under resource constraints," in *Proc. ESEC/FSE*, 2007, pp. 225–234.