Modeling and Engineering Self-Organization in Complex Software Systems

Thesis Proposal

by

Paul L. Snyder

March 2012
Abstract
Modeling and Engineering Self-Organization in Complex Software Systems

Paul L. Snyder
Advisor: Giuseppe Valetto, PhD

Describing, understanding, and modeling the emergent behavior of self-organizing software systems remains an open challenge. Models are key to the systematic engineering of software with self-organizing behavior, which can solve problems in several computing domains where traditional, centralized models are impractical or problematic. These domains include ubiquitous and pervasive computing, peer-to-peer networks, large-scale grids, and Ultra-Large-Scale Systems. Self-organizing approaches have demonstrated great promise in building adaptive behavior and robustness into systems in a decentralized fashion, thus enabling cooperative, but autonomous self-management, and the exploitation of heterogeneity and specificity of system components.

To model self-organization in software systems, one important—and difficult—aspect is capturing the properties of the micro-macro linkage that connects local behaviors of individual system components to global, emergent properties of the system as a whole. In that respect, the design of self-organizing software requires tools beyond those offered by traditional software engineering, and a large number of models have been recently proposed in the literature to address this challenge. These models have been developed separately for different purposes, and they may examine very different features of a system: Descriptive models provide notations that support mainly the design activity and describe mechanisms for engineering self-organizing control mechanisms as reusable building blocks; Validation models allow formal reasoning on the dynamic properties of the system; Analytic models provide techniques for mathematical exploration of abstracted collective behaviors.

The need for a set of tools that integrate the three modeling perspectives above, and enable reusable design, validation, and analysis has been highlighted by our previous experiences building self-organizing software, which have revolved around Myconet. Myconet is an unstructured overlay protocol for peer-to-peer networks, which takes inspiration from fungal growth patterns in order to build an efficient self-optimizing superpeer topology that can also rapidly self-heal in response to damage or attacks. Myconet has proven to be flexible, and we have started to use it as a platform for other self-organizing applications in large-scale distributed systems, including load-balancing in distributed service networks, and detection and mitigation of attacks against the overlay. Those works have demonstrated the applicability of Myconet to a variety of self-organizing scenarios. A new application is created as a Myconet extension, and can be added as a layer on top of the core Myconet overlay. However, to fully leverage some of the Myconet properties, application-level mechanisms may need to interact with the lower-layer Myconet topology management protocol, and may induce alterations to basic overlay behaviors.

Each extension has given us additional insights into the self-organizing dynamics of a system like Myconet, but has also shown us the limitations of ad hoc approaches for the design and analysis of those new applications. These experiences have led us to investigate formal tools and models that may provide the designer of an application-level self-organizing mechanism with early and accurate insight through augmented analytical power. We have examined a number of models and tools from the literature. The goal of this research proposal is to select a small set of synergistic techniques, and build a comprehensive, integrated suite of modeling approaches that cover the design, validation and analysis dimensions. We will use such an integrated suite to model the core Myconet platform and its currently developed extensions, as well as new upcoming extensions, thus experimenting with and demonstrating its features. Once established, this modeling suite can be reused for the principled design of other Myconet extensions, as well as other self-organizing systems. In this way, we will offer a contribution to an increased understanding of how to model and engineer self-organization in software systems.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>4</td>
</tr>
<tr>
<td>2. Related Work</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Classifications</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Descriptive Models</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Validation Models</td>
<td>7</td>
</tr>
<tr>
<td>2.1.3 Analytical Models</td>
<td>7</td>
</tr>
<tr>
<td>3. Background Work</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Myconet and its Fungal etaphor</td>
<td>8</td>
</tr>
<tr>
<td>3.1.1 Myconet Protocol</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Myconet Extensions</td>
<td>12</td>
</tr>
<tr>
<td>3.2.1 Mycoload</td>
<td>12</td>
</tr>
<tr>
<td>Distributed Service Networks Background</td>
<td>13</td>
</tr>
<tr>
<td>The Mycoload application</td>
<td>13</td>
</tr>
<tr>
<td>Extensions to the Myconet protocol</td>
<td>14</td>
</tr>
<tr>
<td>Load-Balancing</td>
<td>16</td>
</tr>
<tr>
<td>3.2.2 HITAP</td>
<td>16</td>
</tr>
<tr>
<td>4. Proposed Work</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Research Plan</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Example Scenario</td>
<td>23</td>
</tr>
<tr>
<td>5. Preliminary Results</td>
<td>23</td>
</tr>
<tr>
<td>5.1 Design Patterns for Self-Organization</td>
<td>24</td>
</tr>
<tr>
<td>5.1.1 Specialization Pattern</td>
<td>24</td>
</tr>
<tr>
<td>5.1.2 Generalized COLLECTIVE SORT Pattern</td>
<td>26</td>
</tr>
<tr>
<td>5.2 Case Study: Modeling HITAP</td>
<td>28</td>
</tr>
<tr>
<td>6. Conclusion</td>
<td>31</td>
</tr>
<tr>
<td>Bibliography</td>
<td>34</td>
</tr>
</tbody>
</table>
Abstract
Modeling and Engineering Self-Organization in Complex Software Systems

Paul L. Snyder
Advisor: Giuseppe Valetto, PhD

Describing, understanding, and modeling the emergent behavior of self-organizing software systems remains an open challenge. Models are key to the systematic engineering of software with self-organizing behavior, which can solve problems in several computing domains where traditional, centralized models are impractical or problematic. These domains include ubiquitous and pervasive computing, peer-to-peer networks, large-scale grids, and Ultra-Large-Scale Systems. Self-organizing approaches have demonstrated great promise in building adaptive behavior and robustness into systems in a decentralized fashion, thus enabling cooperative, but autonomous self-management, and the exploitation of heterogeneity and specificity of system components.

To model self-organization in software system, one important—and difficult—aspect is capturing the properties of the micro-macro linkage that connects local behaviors of individual system components to global, emergent properties of the system as a whole. In that respect, the design of self-organizing software requires tools beyond those offered by traditional software engineering, and a large number of models have been recently proposed in the literature to address this challenge. These models have been developed separately for different purposes, and they may examine very different features of a system: Descriptive models provide notations that support mainly the design activity and describe mechanisms for engineering self-organizing control mechanisms as reusable building blocks; Validation models allow formal reasoning on the dynamic properties of the system; Analytic models provide techniques for mathematical exploration of abstracted collective behaviors.

The need for a set of tools that integrate the three modeling perspectives above, and enable reusable design, validation, and analysis has been highlighted by our previous experiences building self-organizing software, which have revolved around Myconet. Myconet is an unstructured overlay protocol for peer-to-peer networks, which takes inspiration from fungal growth patterns in order to build an efficient self-optimizing superpeer topology that can also rapidly self-heal in response to damage or attacks. Myconet has proven to be flexible, and we have started to use it as a platform for other self-organizing applications in large-scale distributed systems, including load-balancing in distributed service networks, and detection and mitigation of attacks against the overlay. Those works have demonstrated the applicability of Myconet to a variety of self-organizing scenarios. A new application is created as a Myconet extension, and can be added as a layer on top of the core Myconet overlay. However, to fully leverage some of the Myconet properties, application-level mechanisms may need to interact with the lower-layer Myconet topology management protocol, and may induce alterations to basic overlay behaviors.

Each extension has given us additional insights into the self-organizing dynamics of a system like Myconet, but has also shown us the limitations of ad hoc approaches for the design and analysis of those new applications. These experiences have led us to investigate formal tools and models that may provide the designer of an application-level self-organizing mechanism with early and accurate insight through augmented analytical power. We have examined a number of models and tools from the literature. The goal of this research proposal is to select a small set of synergistic techniques, and build a comprehensive, integrated suite of modeling approaches that cover the design, validation and analysis dimensions. We will use such an integrated suite to model the core Myconet platform and its currently developed extensions, as well as new upcoming extensions, thus experimenting with and demonstrating its features. Once established, this modeling suite can be reused for the principled design of other Myconet extensions, as well as other self-organizing systems. In this way, we will offer a contribution to an increased understanding of how to model and engineer self-organization in software systems.
1. Introduction

1.1 Overview

Modern computing environments continue to expand in scale and complexity, challenging traditional approaches used for their engineering and control. These systems place increasing burdens on the limited resources of skilled administrators, calling for the development of new and automated approaches for their design, maintenance and management. Autonomic computing is one paradigm that aims at system self-management, and has the goal of developing adaptive software that can respond to changing conditions and environments without human intervention.

Some branches of autonomic computing have used exogenous control approaches drawn from fields such as control theory. More recently, there has been growing interest in endogenous control, where desirable global system behaviors emerge from the self-organizing interactions of many individual elements. This latter approach is of particular interest in the face of computing systems comprised of huge numbers of components with heterogeneous properties and ownership, high levels of dynamism, and indistinct system boundaries. Systems like these are becoming commonplace in disparate application domains such as ubiquitous and pervasive computing, peer-to-peer networks, computational grids and data centers, and we are witnessing a move toward a new generation of ultra-large scale systems [37]. In these conditions, centralized command and control approaches to self-adaptation and self-management problems (such as self-configuration, optimization, and diagnosis) become impractical.

Studies of naturally occurring self-organization in complex adaptive systems (taken particularly from biology, but also from the social sciences and economics) have proven to be a rich source of inspiration for building complex distributed computing systems. Many biological systems, for instance, from individual cells to the human body, from colonies of insects to complete ecosystems, display properties that are quite similar to those that we might like our computing systems to possess: adaption to changing conditions, resilience to injury or disruption, and the spontaneous, decentralized emergence of a desirable order. Moreover, those various systems achieve such results not by resorting to some form of centralized controller or manager, but through self-organization arising from interactions among the components of the system.

The phenomena observed in natural systems are often useful as high-level metaphors, but specifications with additional levels of formality and detail are needed to turn them into working distributed systems. Levels of fidelity of such a model to the original metaphor may also vary: some systems might be need to be only loosely based on some well-understood natural phenomenon, while others may mimic the behavior of a phenomenon that has not yet been exhaustively modeled in the scientific literature.

Experience shows that these natural self-organization metaphors often lead to computing systems with extreme scalability and robustness. Since self-organizing rules operate in local neighborhoods, an individual computing element in a self-organizing system need only consider a small subset of the total information available in the system at any given time; as a consequence, systems designed using these principles may be able to scale more easily to very large numbers of interacting elements. These systems also show emergent adaptive behaviors, leading to redundancy and self-healing even when the system is perturbed or disrupted in new and unpredictable ways.

Self-organization and emergence are related concepts. Discussions of self-organization highlight the radically decentralized nature of an algorithm, which can achieve increased order (either temporal, spatial, or functional) without external direction. Emergence refers to a two-way linkage between the lower (micro) level interactions between individual system elements, and higher (macro) level properties exhibited by the collective system behavior, which are not explicitly specified in—nor reducible to—those lower-level interactions [11].

As the interplay between the micro- and macro-level is often elusive, there is an inherent difficulty in trying to model, explain or predict the behavior of a self-organizing system. Indeed, subjective surprise on the part of an observer has been proposed as a sort of test for the existence of emergence [42]. An element of this complexity is that the behavior of an individual element is characterized by path dependence, resulting not only from that element’s own rules and history, but potentially by the histories of every other element in that system. These properties make such systems difficult to reason about (even if the local rules are simple)
and new modeling techniques are needed to effectively analyze their behavior.

### 1.2 Motivation

Designers of self-organizing software systems that must operate in real-world circumstances have to ensure that the software satisfy requirements across a wide range of operating conditions. As such, describing, understanding, and modeling the nature and mechanics of the micro-macro linkage in a system is key to engineering self-organizing systems. In particular, designers need to be able to reason about these systems and develop some guarantees on desired global properties, and to show that the system is robust and reliable with respect to those properties across a range of conditions.

Many different properties may be of interest, depending on the system under consideration. For an overlay network, for example, properties such as whether the network is a single connected component, the average node degree, or average path length might be considered. More dynamic properties may be examined, as well, such as the time to reestablish an overlay after a failure, or the overall performance of another algorithm running on top of the overlay itself. Direct calculation of these properties is intractable for systems of any size, and thus various simplified versions of a system may be considered during the process of engineering such a system. Many of these approaches make use of statistical techniques that can provide levels of confidence that a system will actually meet a desired set of properties.

One way to approach reasoning about such systems is through modeling techniques. There has been much recent research in this area, but it remains an open problem. Different models capture different concerns and try to explain different aspects of the problems of engineering self-organization and emergence. Certain approaches provide descriptions that support the design of new systems. Others provide techniques to validate the self-organizing dynamics of a system and generate proofs about them. Others build mathematical models for the analysis of the global system behavior at some abstraction level.

These concerns (design, validation and analysis) are all important modeling dimensions. However, it is not possible at the current state of the art to connect models focusing on those distinct concerns. In order to achieve a comprehensive insightful representation of a self-organizing computing system across those three dimensions, we maintain that a multi-perspective modeling process is needed. A major goal of this research work is to identify and select the best available modeling techniques from each of these families, and integrate them to improve the process of creating of new systems, as well as to support the analysis of existing ones.

The need for a set of tools allowing reusable design, validation, and analysis has been highlighted by our previous experiences building self-organizing systems. Several of these systems have centered around Myconet, an unstructured overlay protocol for peer-to-peer networks. Myconet algorithm takes its inspiration from fungal growth patterns, in order to build an efficient self-optimizing superpeer topology that can also rapidly self-heal in response to damage or attacks.

Myconet has proven to be flexible, and we have started to use it as a platform for other self-organizing applications in large-scale distributed systems, including load-balancing in decentralized service networks, and attack detection and mitigation. These works have demonstrated the applicability of Myconet to a variety of self-organizing scenarios and applications. A new application is created as a Myconet extension, and can be added as a layer on top of the core overlay. However, to fully leverage some of the Myconet properties, application-level mechanisms may need to interact with the lower-layer Myconet topology management protocol, and may induce alterations to the basic overlay behavior. The most difficult task when developing a new extension is often to fully understand the impact of those interactions, and properly account for those repercussions from the beginning of the design and implementation activities.

On the one hand, each extension has thus provided us with additional insights into how we can develop self-organizing dynamics in large-scale distributed systems like Myconet; on the other hand, it has shown us the limitations and perils of ad hoc approaches to design and analysis. These experiences have led us to investigate formal tools and models.

We have examined a number of models and tools from the literature. The goal of this research is to select a small set of synergetic techniques, and develop a comprehensive, integrated suite of modeling approaches that cover the design, validation and analysis dimensions. Armed with that suite, we intend to model Myconet and its extensions (either already developed ones or new and upcoming ones). That way, we will experiment
with and validate its modeling capabilities, and demonstrate how it can be used to acquire early on an accurate insight on how to either design new self-organized software, or approach the maintenance of existing self-organizing features, in the face of novel or changed requirements.

This research will result in a collection of transformations for moving between models in different families, as well as an integrated approach to applying those multiple modeling techniques to the engineering of self-organized systems. It will also provide a demonstration of a principled approach to the design of self-organizing applications using the Myconet platform, which will be instrumental in enabling its wider adoption. This unified modeling of Myconet may also serve as a blueprint or template of how the design and analysis of self-organizing applications in general can be addressed.

2. Related Work

2.1 Classifications

The modeling and design of software that enacts self-organizing control mechanisms requires tools beyond those offered by traditional software engineering. In order to examine how the individual dynamics of individual components are linked to (and induce) the global emergent behavior of the system as a whole, a large number of modeling techniques and formalisms have been proposed in the literature. These models have been developed separately for a set of diverse purposes, and they may examine very different features of a system. We have classified these models into three general families: descriptive models, validation models, and analytic models. These model families—with some relationships (or transitions) among them—are depicted in Figure 2.1.

2.1.1 Descriptive Models

Descriptive models are used to support the design of a self-organizing application and its documentation. They draw on—and extend—existing tools and best practices in use in both the software engineering field (such as UML diagrams and agent-oriented software engineering models), or the complex adaptive systems field. A common approach is to organize the design around a limited set of established building blocks, which capture well-understood self-organization dynamics. This has led to the efforts to catalog design patterns for endogenous control of a software system, and to describe how those patterns can be composed [17]. Other research efforts have proposed descriptive notations that capture different aspects of system dynamics and the linkages between individual components and global properties.

Design patterns for self-organization In software engineering, design patterns are a well-established technique for identifying and cataloging reusable solutions to frequently encountered problems [20]. Researchers and practitioners compile catalogs of design patterns, to collect either best practices for general-purpose design and development, or as solutions that are specific to some application domain within software engineering at large.

Similar efforts are being made to identify design patterns for self-organizing, emergent software, though these are still in the early stages. Babaoglu [1], De Wolf and De Helvoet [13] are two of the earliest works to contribute to this thread of research. Another approach is discussed by Fernandez-Marquez et al. [18], which makes a useful distinction between mid-level patterns (e.g., gradients) and the lower-level patterns that comprise them (in the gradient case, aggregation, spreading and evaporation), as well as the more complex, high-level patterns that provide desirable emergent properties (such as chemotaxis and quorum sensing), and can themselves make use of the mid- and low-level patterns. Starting from this catalog, and based on the lessons learned with Myconet, we have derived an additional new pattern, specialization and an extension of a previously proposed pattern, collective sorting, as discussed [16] and in Chapter 5 below.

The usefulness of a pattern-based approach to modeling and design is that patterns can serve as shrink-wrapped, reusable design elements, that apply a fitting solution every time a desired self-organizing behavior that maps to that solution is required. However, there is generally no leverage for validating or analyzing
the global system behavior from a descriptive model and a system design based patterns. Patterns that hew too closely to their original, inspiring metaphor can also be limiting, resulting in a design that diverges from the semantics of the problem, and in these cases patterns do not provide much help for \textit{ab initio} designs not found in the catalog.

\textbf{Design diagrams} Prevalent notations—including notations that capture design patterns—are only semi-formal, and as such may not provide much support for more rigorous analyses of system properties. A major example is the Unified Modeling Language (UML) \cite{53}. UML is a multi-perspective notation for specifying the design properties of a software system, which uses a multiplicity of diagram . UML is intended for object-oriented design, although some of its notation, such as sequence diagrams and activity diagrams, can also be used in other design paradigms. UML has seen a number of extensions for modeling the design of self-organizing software systems, discussed below.

Models like the ones that can be written in regular UML, or its extensions, are useful to facilitate the development of a self-organizing application through a process of forward engineering (transition DS in Figure \ref{fig:models}). The reverse engineering of a design model from a working self-organizing system (transition SD) is also possible; however, automated reverse engineering tools are in general unable to highlight which design elements are significant to self-organization. In particular, automated identification of self-organizing patterns is an open problem.

Agent-oriented software engineering has also provided several design approaches. Some, like AUML \cite{2}, borrow once again from UML. Other agent-based approaches such as GAIA, MaSE, PASSI, and Prometheus \cite{4} exist. All of these descriptive models focus on the specification of individual elements (agents) of a self-organizing system and of the autonomous low-level behaviors those elements follow, but do not provide a framework for the design of the micro-macro links and the specification of the overall system dynamics.

To that end, one extension proposed by is to augment UML with an explicitly modeled “meso-layer” capturing the control flow dynamics of self-organizing mechanisms using UML state machine diagrams as they unfold in between multiple agents. De Wolf uses UML Activity Diagrams to instead trace the movement of information (data flow) as it is manipulated by elements throughout the system. While informative, these intertwined flow diagrams do not scale well, that is, quickly become quite complex, and are difficult to modularize and compose. \cite{10} Other approaches bring Causal Loop Diagrams (CLDs) from the domain of complex adaptive systems studies into the software design process. CLDs describe feedback loops that exist within the system. Renz and Sudeikat \cite{41,50} extend AUML with CLDs, in order to model the macro-level relationships between system states. Although CLD descriptions are simple and may often be enlightening, they are quite informal and the level of reasoning they enable remain limited.
2.1.2 Validation Models

The goal of validation models is to study the properties of a system, and verify that it fulfills its requirements. The topic of models suitable for validation of SW has been studied extensively in software engineering, but self-organizing applications, with their many-fine grained, still quite complex (and also highly distributed and highly concurrent) interactions, and with their emergent dynamics, pose a great challenge to traditional validation approaches.

Examples of proposed validation techniques that may be suitable in this context include the use of formal languages, such as event calculus and stochastic π-calculus (Sπ). Stochastic π calculus is a formalism for analyzing concurrent systems, which draws on previous applications to biological and chemical systems. This approach models roles and groups, and allows the derivation of quantitative properties of the whole system [51]. In [38], an event calculus description of an observed sociological system is used to model actions of each participating agent over time; this description can be directly analyzed, as well as being compiled into an executable simulation of that self-organized system. Other approaches use established formalizations from software validation research and the analysis of biological systems. For example, Stamatopoulou et al. proposed an hybrid of Communicating X-Machines and Population P Systems [48]. Communicating X-Machines (CXMs), an extension of conventional state machines, have been used extensively in the literature discussing the validation of distributed systems in general. Population P Systems (PPS) are an extension of membrane computing’s P System model that add notions of non-hierarchical reproduction, death, and non-hierarchical linkages between tissues. Thus, CXMs can be used to model the internal rule-structure followed by individual agents, and a PPS can be derived from it in order to model the collective behavior of the system. Some tools exist allowing the programmatic manipulation and visualization of these models.

Validation models can serve as starting points for the initial engineering of a system (transition VS), with guarantees that the implementation will fulfill the desired (validated) properties. Direct derivation of a validation model from an existing system by reverse engineering (transition SV) may also be possible, though this has not been examined much at all in the literature.

2.1.3 Analytical Models

Analytical models offer another approach to understanding self-organizing systems by reducing their dynamics to a set of simplified mathematical equations that often abstract away lower-level details. This approach allows terse descriptions of global behaviors and provides a foothold for the exploration of an application’s parameter space. In the best case, this can enable direct optimization of the system [39].

A number of analytical models use concepts from physics or information theory, and focus on characterizing the statistical complexity of the system, for example by measuring its entropy [30]. These approaches provide tools for assessing the self-organizing properties of a system once it has been developed, but are less helpful for initial system design.

One challenge when working with closed-form equations is that calculating the exact state of large systems quickly become intractable. Thus, approaches for working with analytical models must frequently use statistical methods. An example of this can be found in Parunak and Brueckner, use an entropy model to analyze pheromone coordination mechanisms [55]. From their initial system of equations modeling the system and its entropy measures, they use state partitioning and Monte Carlo simulation to obtain quantitative measures of system behavior.

In some cases, analytical models can serve as an initial design point for software systems, as in [TOCITE:proto], who derive interaction protocols from such a starting point (Transition AS). Techniques have also been explored for deriving a system of differential equations starting from a CLD (transition DA) and from an Sπ-calculus model via reaction equations (transition VA) [41]. This approach allows the create of a simpler continuous model that still reflects important properties of the system under consideration.
3. Background Work

Our investigation of self-organized software has focused on peer-to-peer (P2P) networks, as these systems possess many properties that make them well-suited to self-organizing approaches. Obtaining a global view of the network can be extremely difficult or expensive. They can be extremely dynamic, as peers frequently join and leave (whether by failure or deliberate disconnection), causing “churn”. The number and characteristics of peers taking part in a particular network is unlikely to be known in advance, and no global view or centralized control of the network exists. Therefore, to maintain the network connected and to deploy it in a topology that is suitable for some application that must run on top of it, decentralized protocols that rely only upon local information and actions, while at the same time presenting coherent emergent properties at the global level, are necessary.

Our work in this area, Myconet, develops such a self-organized overlay super-peer based model. Super-peer overlays attempt to exploit the heterogeneous capacities of the participating peers to improve performance for the entire network [33, 29]. Superpeers may take on service roles for other peers, such as indexing files, routing data, or forwarding searches. Connections between superpeers serve to reduce the network diameter and make these services more efficient. This chapter discusses the core Myconet protocol and the two major extensions that have been developed for it: the Mycoload clustering and load-balancing system and the HITAP self-protection mechanism.

3.1 Myconet and Its Fungal Metaphor

Myconet draws its inspiration from some characteristics of fungi. Fungi are much more than mushrooms and yeast, and many species reproduce primarily by vegetative growth; that is, by extending filamentous strands (called hyphae) through the soil or other growth medium. These hyphae search for biomass to assimilate, collecting nutrients and water. The hyphae concentrate the biomass and also use it to fuel hyphal growth. The system of hyphae is referred to as a mycelium. The mycelium constantly adapts to changing environmental conditions by routing nutrients and biomass to areas of need.

Mycelia grow using decentralized, local interactions from which organization emerges, and are able to adjust to changing conditions or damage by dynamically altering hyphal structures. Branching hyphae may join together in a process termed anastamosis, resulting in a multiply-interconnected network that is efficient at transporting nutrients, robust to environmental stresses, and self-healing if injured or disrupted.

Myconet is designed based upon the aforementioned hyphal growth patterns, and aims at showing how they match particularly well with the desiderata for unstructured peer-to-peer network, thus introducing a novel application of the fungal metaphor within the domain of distributed self-adaptive systems.

Myconet uses a relatively simple collection of rules and parameters to regulate the growth and maintenance of the overlay, with each peer continually adjusting its state and connections as needed. Our rules are loosely inspired by a fungal metaphor, in which regular peers are regarded as “biomass” and the superpeers as hyphae criss-crossing the network. We examined several cellular automaton-based models of fungi from the biological literature [3, 31, 24], but opted in the end for a more informal, less constraining design that reflects several dynamics of hyphal growth and achieves the desired properties of emergence and self-organization.

Superpeers in Myconet—which we call “hyphal peers” or “hyphae”—dynamically transition between three states in response to changing network conditions. This adaptive multilevel topology, with its superpeer promotion/demotion logic and semantics, is one of the contributions, and a distinctive characteristic of Myconet. The goal of the transitions between hyphal states is to ensure that load gets pushed towards high-capacity, stable peers, and to build interconnections into the overlay among those peers, in a way that ensures resilience. The state transitions are illustrated in Figure 3.1 and are explained in detail in the rest of this section.

The mission of hyphal peers is, first of all, to concentrate in their neighborhood biomass peers as close as possible to their capacity. Several protocol rules adjust the overlay topology in accordance to heterogeneous peer capacities, in order to balance the growth or contraction of the overlay with the concentration of biomass around those hyphal peers best able to make use of it. In Myconet, each peer is characterized by a capacity value representing the number of biomass peers that it is able to support. Capacity is an abstraction that make
take different semantics, based on application-dependent concerns, for any application that runs on top of the overlay. In a practical implementation, capacity may map onto factors that indicate peer resources and capabilities, such as bandwidth, computational power, or other similar attributes. Peers may be over capacity, but only temporarily; in such a case our protocol will shift biomass peers away from those hyphae (while possibly promoting other peers to hyphal status, as discussed below).

Besides attracting and concentrating biomass peers, each hyphal peer maintains a configurable number \( C_n \) of links to other hyphal peers, which we call “hyphal links”. The total neighborhood of a hyphal peer is, therefore, the collection of biomass peers that are connected to it, plus the other hyphal peers with which it maintains a link. It is well-known that a strongly interconnected network of superpeers improves the efficiency of certain operations performed over a P2P network, such as search \([57, 21, 58]\). Furthermore, it also strengthens the fabric of the overlay in the event of the superpeer failure \([52]\). In addition to those general benefits, in Myconet hyphal links also provide the means for hyphal peers to exchange biomass peers with one another, driving the overlay towards higher level of utilization of the hyphal peers’ capacity. Hyphal links are represented in our system as direct pointers between hyphal peers.

When hyphal peers have achieved \( C_n \) interconnections among one another, they will stop creating additional hyphal links unless they again fall below that level. (This may happen because of protocol operations or because a neighbor hypha leaves the network due to churn or failure.)

### 3.1.1 Myconet Protocol

Bootstrapping of a Myconet network requires a way for peers to discover other live biomass and hyphal peers. We follow the approach used by (among others) SG-1 \([25]\) and use a gossip mechanism to allow peers to acquire information about other peers. PeerSim framework provides a simplified Newscast protocol, which enables each peer to pick randomly another peer, and exchange with it peer lists and state information, that is, whether a peer is biomass, extending, branching, or immobile. Myconet leverages this lower-level protocol
when it needs to select an arbitrary, non-neighbor hypha.

**Biomass Peers**

When protocol execution begins, all peers are in the biomass state and there are no hyphal peers. Any single disconnected biomass node \( b \) will first try to find a suitable hypha to connect to, by querying the lower-level Newscast protocol (Transition 1 in Figure 3.1). \( b \) will consider only hyphae with available capacity, but it won’t otherwise discriminate, choosing randomly in case of multiple possible options. In our current implementation, \( b \) attempts to attach to a single hyphal peer. Several works on P2P networks add an additional level of resilience by allowing leaf peers to attach to multiple superpeers (e.g., [32, 29]). Myconet does not currently allows for that, as it rather relies primarily on its protocol rules to achieve robustness in the network topology. Therefore, once the biomass peer \( b \) gets attached to a hyphal peer, it takes no other action.

In case no hypha is found, the biomass peer will “spore” (Transition 2 in Figure 3.1), becoming a standalone, extending hyphal peer, which will then be able to attract other neighboring biomass peers, as well as to connect to other hyphae.

**Hyphal Peers**

Hyphal peers (i.e., Myconet superpeers) may be in one of three states: *extending*, *branching*, or *immobile*. In accordance with the fungal metaphor, extending hyphae are those that continuously explore the network, foraging for new biomass. (It must be noticed that, as just described, it is actually the biomass peers that seek out an extending hypha and attach themselves to it, but the result is equivalent.) Maintaining some hyphal peers at the extending state at all times allows the network to incorporate new peers into the overlay.

Branching hyphal peers are responsible for growing new extending hyphae and building interconnecting links to other hyphal peers. Immobile hyphal peers are those that are at or near full utilization and have achieved the ideal number of hyphal links (\( C_n \)); they pull biomass from branching and extending hyphal peers to keep themselves at full capacity, and regrow hyphal connections whenever they are lost due to churn or failures.

As the protocol continues to execute, the “highest-quality” peers (in our simulation model, those with the highest capacity) will converge towards, and ultimately reach, the immobile state. The protocol guides the overlay toward that goal by using the following general rule: A hyphal peer \( h \) always looks for the highest-capacity biomass peer \( b_{\text{max}} \) that is either a direct client of \( h \), or a client of one of \( h \)’s neighbor hyphae. If \( b_{\text{max}} \) has a capacity higher than \( h \) itself, then \( b_{\text{max}} \) and \( h \) swap roles; \( b_{\text{max}} \) becomes a hypha, all of \( h \)’s biomass peers and links to other hyphae are transferred to \( b_{\text{max}} \), and \( h \) reverts to being a biomass peer attached to \( b_{\text{max}} \). This rule progressively promotes the highest-capacity peers to hyphal peer status.

Hyphal peers then follow further rules depending on their current state: extending, branching, or immobile.

**Extending hyphal peers**

In the case in which an extending hypha \( h_e \) is not connected to any branching or immobile hypha, \( h_e \) will attempt to form a hyphal link to a random peer of one of those types (random peers are selected by querying the lower-level Newscast protocol). These links ensure that the initial, stand-alone clusters that form around extending hyphae in the bootstrapping stage of the protocol will gradually converge towards a single, larger connected network.

If no immobile or branching peers can be found, \( h_e \) will then try to connect to another extending hypha \( h_{e2} \). Whenever two extending hyphal peers have become neighbors (whether because of this rule or as a result of other protocol dynamics discussed below), the larger will attempt to “absorb” the smaller (Transition 3 in Figure 3.1), if it has sufficient unutilized capacity. When \( h_e \) absorbs \( h_{e2} \), all of \( h_{e2} \)’s biomass peers are transferred to \( h_e \), all of \( h_{e2} \)’s hyphal links are transferred to \( h_e \), and \( h_{e2} \) reverts to being a biomass peer attached to \( h_e \). This rule is the main means by which the number of superpeers in the network is contracted, once again favoring hyphal peers with higher capacity.
Finally, if the extending hypha has reached or exceeded its biomass capacity, it will become a branching hypha (Transition 5 in Figure 3.1) and the excess capacity will be handled by the rules for branching hyphal peers.

**Branching hyphal peers**

Branching hyphal peers are at or near their ideal biomass capacity but have yet to reach their ideal number of hyphal connections $C_n$ (see above). Their purpose is multifold: they help to regulate the number of extending hyphal peers in the network, expanding or contracting their number to adequately handle any biomass peers; they act as a conduit to move biomass between extending and immobile hyphae; and they seek to construct new inter-hyphal-peer connections, which are important for the robustness of the overlay.

Branching hyphal peers adjust the number of extending hyphae by attempting to maintain one and only one connected extending hypha. If a branching hypha $h_b$ does not have a link to an extending neighbor (as is the case after it is first promoted to branching status), it will choose its highest-capacity biomass peer and promote it to the extending hypha state (Transition 4 in Figure 3.1). Branching hyphal peers also help contract the overall number of hyphae in the network: If a branching hypha $h_b$ is connected to two (or more) extending hyphae, it will pick two of them, $h_{e1}$ and $h_{e2}$, and connect them to each other. $h_b$ maintains its connection to the higher-capacity peer $h_{e1}$, but severs its connection to $h_{e2}$. The rules for extending hyphae will then be triggered, leading to the collapse of those two extending hyphal peers in the following round (Transition 3 in Figure 3.1).

Branching hyphal peers also work to pull biomass from extending hyphal peers; similarly, immobile hyphal peers pull biomass preferentially from branching peers. This way, biomass peers tend to gradually aggregate around those high-capacity hyphal peers that have established themselves in the network over a period of time. Notice that since promotion to branching status occurs only after an extending hypha reaches full utilization, pulling of biomass occurs only when a branching hyphal peer has lost some of its own biomass. That may occur because the biomass has been pulled from them by connected immobile hyphae, or because some biomass peers have left the network.

Whenever a branching hyphal peer has fallen below full utilization, it tries to obtain new biomass from its neighbors. If branching hypha $h_b$ is of larger capacity than a neighbor $h_n$ and $h_b$’s unused capacity is greater than the number of biomass peers attached to $h_n$, $h_b$ will absorb $h_n$ outright (Transition 7 in Figure 3.1). All of $h_n$’s biomass peers attach to $h_b$, all of $h_n$’s hyphal links are transferred to $h_b$, and $h_n$ becomes a biomass peer of $h_b$.

If $h_b$ is still not at full capacity after attempting to absorb neighboring hyphae, it then checks whether any of its connected extending hyphal peers have biomass peers; if they do, it will transfer biomass peers to itself until it is at capacity, or until no more biomass is available. In the latter case, if $h_b$ is still under-utilized, it will drop back down to extending status (Transition 6 in Figure 3.1).

If, as a result of the processes described above, a branching hypha exceeds its capacity (that is, if it has too many connected biomass peers), it will push the excess biomass down to a neighboring extending hypha.

A branching hyphal peer also seeks to reinforce the fabric of the overlay by growing links to other, randomly-selected, existing hyphae: that approximates the process of anastomosis in natural fungi. These cross-connections add resilience to the network in case of the failure of one or more hyphal peers. Whenever a branching hypha has accumulated enough hyphal links such that it reaches or exceeds the parameter $C_n$, it promotes itself to immobile status (Transition 8 in Figure 3.1).

**Immobile hyphal peers**

Immobile hyphal peers have achieved their ideal number of hyphal connections and have connected biomass peers sufficient to saturate their capacity. If an immobile hypha falls under its biomass capacity, it will attempt to absorb biomass from a connected branching or extending hypha. In this way, immobile hyphal peers (which not only are high-capacity, but have also proven to be stable by having successfully transitioned through all stages of our protocol) continuously maintain high levels of utilization, and client biomass peers are moved to these peers in preference to other hyphal peers.
If two extending hyphae are connected to an immobile hypha $h_i$, $h_i$ will create a direct connection between them, so that the lower-capacity hyphal peer is absorbed by the higher-capacity one. (This rule is identical to the rule for branching hyphal peers.) Also, if $h_i$ is connected both to a branching and an extending hyphal peer it will similarly enable the absorption of the extending hypha by the branching hypha.

Next, if an immobile hyphal peer $h_i$ is of larger capacity than a neighbor $h_s$ and $h_i$’s unused capacity is greater than the number of biomass peers attached to $h_s$, $h_i$ will absorb $h_s$ outright (Transition 10 in Figure 3.1). All of $h_s$’s biomass peers become attached to $h_i$, all of $h_s$’s hyphal links are transferred to $h_i$, and $h_s$ becomes a biomass peer of $h_i$.

$h_i$ then checks to see if it is still under its biomass capacity. If it is, it will attempt to absorb biomass from neighbor branching or extending hyphae. $h_i$ transfers biomass from such peers until it has reached full utilization or no further biomass peers are available.

If, after this process, $h_i$ is over its biomass capacity, it will transfer the excess to a neighbor with available capacity (immobile, branching, or extending, in order of preference). If $h_i$ does not have any under-capacity neighboring hyphal peers, it will promote its highest-capacity biomass peer to extending status, and transfer the excess capacity to it (Transition 4 in Figure 3.1).

If $h_i$ drops below the number of hyphal links $C_n$ that Myconet is trying to maintain, whether because any of the rules described above, or because of peers leaving the network, it will randomly form new links to another existing hyphal peer. (As with branching hyphal peers, these are selected from the list maintained by Newscast.) Also, to prevent the topology from becoming stagnant, there is a small probability $p$ that $h_i$ may form another hyphal connection even if it already has $C_n$ neighbor hyphae. (In our simulations, we used $p = 0.05$.)

If, because of any of the above rules (or because other hyphal peers have chosen to connect to it), $h_i$ has more than $C_n$ connections to other hyphae, it will randomly drop extra hyphal connections until it gets back at the $C_n$ level.

Finally, if $h_i$ falls below a certain utilization threshold $u_i$ and is unable to regain the lost biomass, (in our experiments we set the threshold equal to 80% of $h_i$’s capacity), it will demote itself to become a branching hypha (Transition 9 in Figure 3.1).

**Results** The Myconet protocol effectively constructs and maintains a strongly interconnected, decentralized superpeer overlay that scales to at least 106 peers. This overlay quickly converges to an optimal number of superpeers and high levels of capacity utilization. Myconet’s greatest strength is its ability to self-heal the interconnected overlay, quickly repairing the damage from catastrophic events in which 30-50 removed from the network. Even when 80% capacity, most stable superpeers are removed, Myconet is able to quickly reconfigure its overlay to reflect the new capacity of the network while dynamically adjusting the interconnections between superpeers. A detailed discussion of experimental results with the core Myconet platform can be found in [47].

3.2 Myconet Extensions

The overlay maintenance approach described so far represents an example of a large-scale self-organized software, which, in this case, builds a distributed computing infrastructure with certain desirable properties, like resilience, self-healing, and efficient organization of resources according to the heterogeneous capacity of its elements (peers). On top of such an infrastructure, it is possible to develop a layer of applications that leverage those properties, and carry out specific tasks. We discuss here two such applications, which we name “Myconet extensions”: Mycoload and HITAP.

3.2.1 Mycoload

Mycoload is an extension that deals with the construction of a decentralized service network, built as an application on top of the Myconet overlay.
Distributed Service Networks Background

The provisioning of distributed computational services has followed a constant trend towards increasing degrees of virtualization. The original client/server model led to the evolution of dedicated clusters, followed by grids, and then by large-scale data centers able to host a multitude of diverse applications on virtual machines. That trajectory has now led to cloud computing, with its promise of opaque, elastic, and on-demand allocation of computational resources, and its computing-as-a-utility model.

Nowadays, cloud computing environments are typically quite large and complex, and under the centralized ownership and administration of a single entity, which provisions the resources, the networking and software infrastructure. However, there are also early signs of a further virtualization stage: a move away from centralized clouds, and towards decentralized service networks, in which the resources are not concentrated under a single ownership, but can be located anywhere on the network and administered by multiple organizations (or even individuals), and are engaged on the fly to satisfy clients’ requests. This model borrows some traits from the idea of Edge Computing\cite{9,15}, but expands it in a couple of new ways. First of all, it does not assume any hierarchical relationship between the various entities at the network’s edge that put their resources together, in contrast with the classic Edge Computing model, which distinguishes between a primary origin location and one or more supplementary edge locations for resources; moreover, it is not limited to the planned deployment of a set of pre-defined services on well-chosen host nodes, but it embraces the SaaS (Software-as-a-Service) idea of the cloud, which strives to accommodate unanticipated and diverse client computing needs.

Decentralized service networks are currently being pioneered mainly by industry. Research in this area is so far limited to a few, very recent, contributions (including \cite{8,40}), although this service provisioning model offers – side by side with its inherent promises – several interesting challenges. For example, the topology of a decentralized service network changes all the time, with participating nodes leaving and joining without notice or control, resulting in churn patterns that may be very different from what is expected in a regular cloud. Also, to build and operate a decentralized service network, one needs the ability to direct incoming requests to nodes that host services able to satisfy those requests, and, to effectively share the load among those nodes. To achieve that, one option is an entity that serves as a gateway or intermediary, accepting requests and re-directing them as needed to the appropriate nodes. That implies, however, that such an intermediary must maintain accurate and up-to-date information on where all services are located in the decentralized service network. That may become impractical as the scale of system increases.

The Mycoload application

In Mycoload, peer capacity indicates that peer’s processing power: higher capacities represent better suitability provide a service, that is fulfill service requests that are routed to that peer from service clients or other peers in the network. Mycoload also introduces the concept of a service type for peers: each peer is characterized by a single service that it is able to provide.

Mycoload extends the Myconet protocol with rules that aim at constructing clusters of peers offering the same service type. Then, self-organized load balancing is applied to the peers within each cluster, to efficiently distribute requests for services of each type among the peers’ job queues, following a variant of the algorithm in \cite{18}).

Load balancing is performed between same service-type neighbors within a cluster, with local exchanges striving to keep queue lengths proportional to the capacity of each peer. When a load-balancing operation is performed, a peer selects a random same-type superpeer neighbor; as biomass peers have only a single neighbor, they will be the ones to initiate the load-balancing operations with their parent. This way, load-balancing operations tend to occur preferentially between the more-capable superpeers, which helps ensure that jobs will be balanced throughout the cluster. “Ideal” queue lengths are calculated by determining the total number of jobs in both queues and dividing the jobs proportionally based on the peers’ capacities.
Extensions to the Myconet protocol

We have extended Myconet in order to make it suitable for the Mycoload decentralized service network application. New rules support the clustering of same-type peers. On top of that, the self-organized load-balancing is applied to the requests for the various service types that entering the decentralized service network that is deployed onto the overlay.

Myconet is good at consistently selecting the highest-capacity and most stable peers to serve as its superpeers. The first application-specific design decision we made for Mycoload was therefore to map the capacity of a peer to the amount of service requests that peer is able to fulfill in a time unit. This, in turn, is reflected in the Myconet overlay as the target number $B_S$ of client peers maintained by a superpeer, as we want higher-capacity peers to connect to a large number of lower-capacity same-type neighbors. This way, superpeers are well positioned to efficiently balance their neighbors’ request queues.

Moreover, as Myconet had originally been developed to manage peers irrespective of service type, another extension to the protocol was to make it aware of peer types. A Mycoload peer may have at any given time both same-type and different-type neighbors, but, as discussed, a peer can only perform load-balancing operations with same-type peers. Therefore different types of rules apply when that peer manages its links to same-type vs. different-type neighbors, to ensure the incremental construction of neighborhoods in the overlay, in which one or more hyphal peers aggregate and serve a number of biomass peers of the same type.

The current implementation assumes that peers provide only one service type, and hence participate in only one cluster. Under this model, peers could offer more than one service type by running multiple instances of the protocol. Here we present results for single-type peers only; extending Mycoload to allow peer to offer multiple service types is planned as future work.

Cluster Construction Rules  To enable the clustering of same-type peers, Mycoload uses a set of rules that regulate the interactions of same-type vs. different-type peers. We focus here on those rules, and mostly gloss over the mechanics of superpeer selection, promotion and demotion, which remains the same as in the Myconet basic protocol described above.

Biomass Peers  All peers begin protocol execution as biomass. A disconnected biomass peer $b$ will attempt to connect to an extending superpeer it can find through the lower-level gossip protocol. However, it will only select one of its same type. If no suitable extending superpeer can be located, $b$ will promote itself to extending status. This rule ensures that there are always extending hyphae of each type in the network, since these self-selected extending peers must act as aggregation points for isolated peers of their type. If a biomass peer ever becomes disconnected, it searches for a new extending hypha to connect to, as during bootstrapping.
Hyphal Peers - All States  Hyphal peers aggregate biomass peers of their same type up to a target level $B_S$ (proportional to their capacity); to that end, hyphal peers in a higher protocol state will always try to “pull” biomass from other neighboring hyphal peers of their type in lower states, or in the same state but of lower capacity, through the execution of absorption rules.

Hyphal peers also form the backbone of the overlay, and are responsible for its robustness; to that end, hyphal peers attempt to create and maintain a target number of links $C_S$ to other hyphae of their type, which is a parameter of the protocol. In Mycoload, there is another parameter, as hyphal peers also try to maintain $C_O$ links to hyphae of different types. $C_O$ ensures that clusters do not become disconnected from one another.

Increasing the $C_S$ or $C_O$ parameters increases the number of cross-connections (and the size of each peer’s neighborhood views), while adding overhead to the protocol. $C_O$ ensures that clusters do not become disconnected from one another.

Hyphal peers act as type matchmakers for other peers; if a hypha $h$ has a neighbor $n$ of a different type, $h$ will check if some other of its neighbors $s$ is of the same type as $n$. If so, $h$ will transfer $n$ to become a neighbor of $s$. If not, $h$ tries the same thing with the neighbors of all of its own neighbors. $h$ will not necessarily drop its link to $n$, if it remains within the limit of its target $C_O$ different-type hyphal links.

Hyphal peers also execute rules specific to their protocol state, as described below.

Extending Peers  Superpeers remain in the extending state until they have reached their target number of biomass neighbors of the same type $B_S$, at which point they promote to branching status. Extending peers also anchor themselves to the overlay with a single link to either a branching or an immobile peer of any type. In the event two extending peers of the same type become neighbors, the lower-capacity peer will transfer all of its biomass peers to the higher-capacity and demote itself to biomass status. This is an absorption rule.

Branching Peers  One primary function of these superpeers is to grow new links to other superpeers, thus building robust cross-connections for the overlay. Branching peers are those that have aggregated their target number of same-type biomass peer ($B_S$), but have not yet reached their targets of $C_S$ and $C_O$ hyphal links to other superpeers of the same type and different types, respectively.

With respect to same-type hyphal links, if a branching peer $h_b$ is under the target number $C_S$, it will search its neighborhood to determine if any of its neighbors are attached to a suitable same-type hyphal peer that is not already its own neighbor. With respect to different-type hyphal links, if $h_b$ is under its target $C_O$, it will randomly select a peer from the gossip cache and grow a neighbor link to it. The resulting peer might be of any type (including the same). A branching peer also attempts to always have one extending peer among its neighbors; if no such peer exists, it will pick its largest capacity biomass child and promote it, obtaining an extending peer of its same type.

If a branching peer ever gets over its biomass capacity, it will push excess biomass children to an attached same-type superpeer. Once it has grown $C_S$ and $C_O$ hyphal links, a branching peer will promote to immobile state.

Immobile Peers  Once a peer has achieved immobile status, it will attempt to maintain it through absorption, that is, by pulling biomass from lower-state hyphae of its same type (possibly resulting in their demotion). Similarly, it will try to grow new hyphal links if some are lost, in order to maintain the target $C_S$ and $C_O$. If the immobile peer, instead, happens to have more than $C_S$ hyphal links to same-type superpeers, or or more than $C_O$ hyphal links to different-type superpeers, it will randomly drop the excess links.

An immobile peer should be connected to either zero or one same-type extending peers. In case it has multiple extending neighbors hyphae, it will connect them together, thus triggering the absorption rule. If it has both branching and extending neighbors, it will transfer the extending peers to become children of the branching peers. In order to prevent the overlay from settling into a local optimum, immobile peers will occasionally grow a new random link by picking a random hypha from the gossip cache and connecting to it.

The addition of the rules described above to the basic Myconet protocol enable Mycoload to quickly settle into an overlay configuration where peers of each type are concentrated around, and connected to, one or more hyphal peer of that type. Figure 3.2 shows with color codes how such a self-organizing clustering occurs. The first snapshot shows the overlay during bootstrapping phase (at round number 4, when small
clusters are beginning to self-aggregate), and the second snapshot shows the overlay after it has stabilized its topology at round 14. These figures were taken from a Mycoload run with 150 peers, 5 types, $C_S = 2$, $C_O = 2$, and max peer capacity of 15.

### Load-Balancing

Load-balancing is performed between neighbors. In Mycoload, because of the heterogenous capacity of peers, the goal of load-balancing is to make queue lengths proportional to the capacity of each peer.

Based on this principle, when a load-balancing operation is performed, a peer $p_a$ selects a random same-type hyphal neighbor $p_b$. If $p_a$ is a biomass peer, it will have only a single neighbor, a same-type hypha. If $p_a$ is a hyphal peer, it will only try to balance its queue with other same-type hyphal peers; as biomass peers have only a single neighbor, they will be the ones to initiate the load-balancing operations with their parent. This way, load-balancing operations tend to occur preferentially between the more capable superpeers, which helps ensuring that jobs will be balanced throughout the cluster. $p_a$ and $p_b$ compare their queue lengths; “Ideal” queue lengths are calculated by determining the total number of jobs in both queues and dividing the jobs proportionally based on $p_a$ and $p_b$’s capacities. If the queues are unbalanced, jobs are transferred from the queue that is over its ideal length to the queue that is under that length.

During load-balancing operations, the transfer of a job involves only a transfer of a reference to the job, that is, to the location where that job can be actually retrieved when a peer is ready to execute it. The choice of forwarding only references helps to keep the network overhead as small as possible, since jobs could be composed of large amounts of data.

### Results

Our experimental results for Mycoload show the benefits and the limits of its approach. In particular, Mycoload’s strengths are the reduced number of messages and increased convergence rate with respect to previous work [16], which achieved a maximum network throughput (processed jobs per time unit) of 90% of optimal. Myconet’s results showed a comparable throughput percentage was reached after just 145 messages per peer, and that optimal throughput (100%) was achieved after 529 messages per peer. We attribute this improvement to the more rational structure of the network topology that considers key properties of the peers, particularly their capacity. Myconet’s convergence is somewhat slower the churn is very large, and the experiments showed a reduced ability to optimize the response time when the network is subject to only a light load. For a full discussion of these results, refer to [54].

#### 3.2.2 HITAP

Another Myconet extension is a self-organized approach to detect attacks to the overlay topology that target superpeers, which we call Hormone-Inspired Topology Adaptation Protection (HITAP). HITAP is itself biologically inspired, and is based on the release, diffusion and processing of an “alert hormone” in the network in response to node failures.

While superpeer overlays are resilient to random node failures compared to a non-hierarchical overlay where all nodes have similar degrees [57], they are inherently vulnerable to directed attacks. That is, by specifically targeting superpeers, an attacker can quickly disrupt the whole network, possibly breaking it into multiple components and isolating many of the regular peers. To combat this kind of attack strategy, researchers have begun to investigate self-organizing superpeer protocols, which support the rapid reconstruction of an efficient overlay topology after a disruptive event (either malicious or accidental).

While self-healing protocols can help a network recover quickly from targeted attacks, they cannot directly thwart them. One possible strategy, as suggested among others by Keyani et al. [28] and Zweig and Zimmerman [60], is to implement a provision in the network that causes nodes to switch from a superpeer, scale-free mode, to a “flat” mode, in which the network nodes maintain a uniform or narrower degree distribution. Zweig and Zimmerman [60] discuss how such provision should be reactive and fully decentralized, should rely only on information available locally to each node, and should not depend on an accurate distinction between the failure of a node due to an actual attack, as opposed to an accident. Keyani et al. [28] note that a critical property of such an attack-thwarting provision is the ability to switch back reliably to the more
efficient scale-free topology if the network is subject only to random failures due to normal levels of node churn.

In HITAP, we defend against targeted attacks using an alert hormone to induce modifications of the topology across the whole overlay. The alert hormone propagates whenever a peer (or node) disappears from the network, starting with the neighbors of the failed node, and throughout the network. The local concentration of the hormone at each node is used as a hint that an attack may be happening somewhere in the overlay. When the concentration at a node exceeds a threshold, it induces the node to make a local decision to switch from a superpeer-oriented strategy for the overlay construction and maintenance, to a strategy in which the node seeks to remain attached to a constant, small number of peers. The diffusion of the hormone in a network under a targeted attack causes the network as a whole to adapt its topology and quickly switch to a “flat” mode, where all nodes have an almost uniform degree. In that mode, an attacker is deprived of major targets, and the network becomes harder to disrupt.

The hormone gets also progressively metabolized by each node. As the attack subsides (or is reduced to simply killing random low-degree nodes) the local concentration decreases; when it falls below a threshold, nodes will switch their protocol back to normal operation, and start working towards re-establishing a superpeer overlay.

**Modifications to Protocol**  Damage to the overlay caused by peer exiting the network (whether by attack or normal churn) results in the generation of a certain amount of alert hormone, which spreads from neighbor to neighbor by diffusion. Once sufficient amounts of alert hormone have accumulated in a peer, it switches into a new protocol state. This state is termed *bulwark*, indicating that the peer has switched into a defensive posture and is acting to reduce the network’s vulnerability to attack. Bulwark peers do not follow the regular Myconet overlay maintenance protocol, but simply work to configure their local neighbor relationships into a flat (non-superpeer) network by maintaining a fixed, small number of connections to other peers. When a peer switches into the bulwark state, it severs connections with all its existing neighbors, causing the generation of yet more alert hormone.

**Alert Hormone Generation**  When a peer fails, the exact amount of hormone generated by the remaining peers is determined by the failed peer’s degree (that is, its number of neighbors). A peer that observes the sudden loss of a neighbor will generate a quantity of alert hormone according to a quadratic relationship, $0.5d^2 - 2d$, where $d$ is the degree of the failed neighbor. This function weights heavily the failure of large degree peers (such as immobile or branching hyphal peers), whereas it is relatively insensitive to the failure of low-degree peers (such as biomass and peripheral extending hyphal peers). Since every neighbor of a failed peer will generate the alert hormone according to this formula, the failure of large-degree peers will result in the release of large amounts of hormone throughout the network.

**Hormone Dynamics**  Diffusion occurs through simple neighborhood equalization. If a peer $p$’s neighbor peer $n$ has a lower hormone concentration than $p$, the two nodes will balance their hormone levels. Over time, this results in the hormone diffusing through the network.

The maximum quantity of alert hormone that may be held at a node is capped, to prevent the build-up of network pockets with very large amounts of hormone. If, after diffusion, the amount of hormone is over this maximum, the excess is discharged by the peer and removed from the system.

Additionally, peers metabolize the alert hormone over time. The local concentration is periodically reduced by a fixed percentage. Thus, hormone levels will spike following a failure, but will slowly drop back down if additional failures do not continue to generate additional alert signals.

**Switching the Mode of the Network**  A peer whose level of alert hormone is over a certain threshold for a set period of time, may switch into the bulwark state.

When a peer makes the jump into the bulwark state, it will drop all connections with its current neighbors. This drop is abrupt, causing all the former neighbors to release alert hormone into the network on their own. The dropping of previous neighbor peers when switching to the bulwark state speeds the spread of alert hormone throughout the network, and may start a cascade of more peers transitioning to the bulwark state,
Figure 3.3: Hormone-based attack detection in an example 150-node network with node capacities assigned via a power-law distribution and a maximum capacity of 20 possible. (a) Two cycles into an attack, some local buildup of alert hormone is occurring. (b) Four cycles into the attack, nodes have passed the transition threshold (red) and switched into bulwark (octagons), causing more nodes to make the jump. (c) Alert hormone levels continue to rise, diffusing through all nodes. (d) Six cycles into the attack, all nodes have switched to the bulwark state. (e) Four rounds after the attack ends, hormone concentrations in most nodes have decayed below the reversion threshold; nodes delay their return by five cycles to reduce thrashing. (f) Nine cycles after the attack ends, alert hormone levels have decayed and the nodes revert to normal operation. (g) Within six more cycles, the nodes have reconstructed the superpeer overlay.

and accelerates the formation of the new, “flat” topology, since cascaded bulwark nodes will connect to a different set of neighbors. See Figure 3.3 for a visualization of protocol dynamics.

When switching, a bulwark peer caches its previous Myconet protocol state. It immediately begins to connect to new neighbors, growing new connections to other peers until it has reached a target number. If all peers enter the bulwark state, the network will quickly converge to a relatively flat topology, without superpeers. The network thus becomes resilient against targeted attacks, although, while in this mode, the system as a whole decreases its efficiency, since it won’t be able to exploit effectively exploit heterogeneous node capabilities.

Once the network has shifted to such a flat configuration, attacks against particular peers are practically indistinguishable from ordinary churn. Because of this, the failure of a bulwark peer does not cause additional alert hormone to be generated.

A peer will remain in the bulwark state until its local concentration of alert hormone has dropped below a threshold value, plus an additional period of time. This latency is a customizable parameter, and is designed to prevent the network from attempting to switch between modes too quickly, which may result in thrashing. The system should try to avoid the overhead of restoring the superpeer topology while waves of hormones are still traversing the network, possibly because an attacker is still trying to disrupt the overlay in some areas.

**Modifications to Other Protocol States**

Besides the addition of the bulwark state, other states of the Myconet protocol were modified for HITAP.

When a peer drops out of bulwark state, it first attempts to return to the Myconet state it had cached when it switched. This provides a “hint” as the network rebuilds, speeding the identification and promotion of high-quality nodes.

Also, bulwark peers are not considered to be hyphal peers, although they can serve as connection points for new peers entering the network (as extending peers do). They are also regarded as biomass when determining a hyphal peer’s utilization.
Thus, any node executing the Myconet protocol may have additional neighbors in the bulwark state, above the number of neighbors that it would normally attempt to maintain; this is particularly significant for biomass nodes, which normally have only a single superpeer connection to its parent, but are allowed to maintain connections to some node in the bulwark state. This is likely to happen only during transitions between overlay modes.

Finally, new peers entering the network will “mirror” the mode of operation of their connection point: a new node will run in either the normal Myconet mode (in which case it will start in the biomass state), or bulwark mode. The new peer will also mirror the concentration of alert hormone of the first peer it gets attached to; this ensures that the new peers have a head start on assessing the current state of the network without, and that they do not act as a net drain on the levels of alert hormone already in the system.

When designing a self-organizing algorithm, a significant portion of the work often revolves around designing a parameter space and choosing appropriate values that support the desired convergent behavior. Correct values for those parameters may not be absolute, but rather may depend on other properties and environmental conditions. In the case of HITAP’s attack detection and thwarting strategy, factors such as the environmental conditions (e.g., churn levels) and characteristics of the peers and the network as a whole have significant effects. A full discussion of these considerations can be found in [45].

Results  Experimental results for HITAP show the efficacy of a topology adaptation defence against targeted attacks. The dynamics of hormone diffusion, concentration and metabolization can be adjusted to ensure that the network as a whole distinguishes attacks from regular peer churn, and transitions quickly and accurately between those two topologies, though without online adaptation the choice of parameter settings determines the system’s sensitivity background network conditions and attack scenarios. A complete discussion of HITAP’s performance under a range of conditions can be found in [45].

4. Proposed Work

A continuing challenge in the design and implementation of self-organizing systems is the understanding of global emergent behaviors and properties that result from rules followed by lower-level components and their self-organizing interactions. When engineering a self-organizing system, the design of desired emergent behaviors is a particular challenge. The result of a particular choice of local rules for self-organizing elements can have unpredictable effects. In general, different models can be used to capture different aspects of a system; each modeling approach evidently has its own strong suit, but also carries some limitations.

Our research will investigate the integration of different models in order to enhance their capabilities for use during system engineering. Even though there have been many recent publications on the modeling of self-organization, there has been relatively little research that looks to bring these these approaches together in a way that supports system engineering. In particular, we aim to develop extensions to these models in order to enable inter-model transitions, resulting in a toolkit for creating and analyzing self-organizing, distributed software systems.

As discussed in Chapter[2] models for engineering self-organizing autonomic systems can be classified into three general families, descriptive models, validation models, and analytic models. Models of different types capture disparate properties of the system under design or examination, and research is needed to develop a unified approach to utilizing these models in the context of a principled engineering process.

- **Descriptive models** are constructive tools that provide support for the design of new systems, but they do not offer much leverage for formal or quantitative reasoning about their properties. Two aspects of descriptive modeling are (1) the capture and modularization of individual self-organizing mechanisms from well-studied metaphors, along with their identified micro-macro level linkages and resultant emergent properties; (2) visual tools for diagramming interactions between elements of a system, flows of information between them, and resulting global behaviors that may not be contained in typical low-level models.
• **Validation models** can be used to prove properties of a particular system, but are less useful to generate design abstractions that can represent valid and general building blocks to guide the design process.

• **Analytical models** support the mathematical exploration of the behavior of actual systems, for example to make sense and refine their parameter space for optimization purposes.

In some cases, is possible to translate between specific models belonging to different families in order to gain the benefits of each, and some of these translations have been examined in the research literature. For instance, it may be possible to produce a validation model starting from a descriptive one, as in Sudeikat *et al.* [51], which offers a procedure for deriving a (Sp) model from a descriptive causal loop diagram (transition DV in the figure). Along similar lines, Renz *et al.*, [41] propose a method for moving from a validation model (specified in Sp calculus) to an analytic model in the form of a system of differential rate equations (transition VA).

Transition in the opposite direction is problematic, as the higher level of abstraction of analytic models results in loss of formal detail about the system. Moving directly from a descriptive to an analytical model may be possible, although the literature only suggests preliminary steps. For example, the specifications of self-organization design patterns may be decorated with analytic annotations about its behavior, as discussed in [17].

Direct generation of a descriptive model from an analytic model (transition AD) is difficult, for the same reasons as transition AV: analytical models tend to abstract out a lot of detail. Since analytical models of biological phenomena (or other Complex Adaptive Systems) developed in the natural sciences are often used as the starting points for the design of new computing systems, it is typical to try to capture them by hand as thoroughly as possible within a suitable descriptive model.

While some works that demonstrate transitions between specific models in different families exist, they represent point solutions, and this problem is in general currently outstanding. Some recent research has also focused on how to move beyond an ad hoc approach to the design of self-organizing systems, and specify a methodology. Several design methodologies have been proposed, typically beginning with a descriptive model of a proposed system (e.g., [12]); however, these methodologies do not directly address the roles of models from different families in their design process.

The issue of a more unified and comprehensive approach to the specification of self-organized systems is therefore an open area of investigation. The objective of our research is to understand how existing modeling techniques can be connected and integrated, when necessary by providing some extensions and bridges among them, so that we can solve in an integrated way the problems of sound construction, production of proofs and guarantees, and dynamics analysis, in the challenging domain of self-organized autonomic software systems. Such integration would not only represent an enhancement that may facilitate the work of researchers and practitioners involved in the development of those systems; it is also likely to lead to new insights on how different concerns interact and intertwine, and impact the creation of emergent mechanisms and self-organized dynamics.

### 4.1 Research Plan

**Stage 1:** Initially we will select from among the many proposed modeling approaches to create a toolkit of the best-of-breed modeling techniques from each of the three families. For this stage, we have identified several approaches with high potential.

• **Layered Design Patterns:** The pattern catalog approach attempts to organizing the growing body of knowledge for engineered self-organization, making them available for reuse by system designers. Of particular interest is [18], which proposes a classification of a number of such mechanisms into three layers: **Basic Patterns**, **Composed Patterns**, and **High-Level Patterns**. (Figure 4.1) This classification also defined composition/usage relations among the corresponding self-organization mechanisms: patterns in the lower layers provide building blocks for more sophisticated patterns at the higher layers. The ability to identify patterns and their role within the context of a self-organizing system at large provides leverage for the design phase of engineering such a system: each pattern represents a well-
understood piece of dynamics with its own micro-/macro-level linkages. Since simple changes to systems with emergent behavior can result in major changes in the overall dynamics, the modularization of the system also improves the level of support for the maintenance and evolution of the system over time as changes can be more easily targeted to contained areas of functionality.

- **Causal Loop Diagrams**: The literature of complex systems frequently uses Causal Loop Diagrams (CLDs) as a tool for analyzing patterns of interaction \[49\]. They have recently been used by Sudeikat and Renz to analyze multi-agent systems, deriving the causal links by tracing both the internal reasoning and external interactions of the individual elements \[51\].

Unlike low-level approaches, CLDs are a high-level tool that focuses on global properties; in particular, a key goal is the identification of positive and negative feedback loops among different variables. By determining how a change in one variable results in a change in another, and tracing these chains of causation, the identified linkages between balancing and reinforcing mechanisms can give insight into the overall dynamics of a system. This can be useful when working to modularize a system’s design, as well as indicating which areas of a system are likely to be affected by a change (and, hence, which will be likely to also require further modification and tuning).

- **Communicating X-Machines with Population P Systems**: While patterns can provide starting points for designing complex emergent behaviors, additional tools are required to understand the higher-level dynamics that result from the composition of the identified lower-level mechanisms and the effects of adjusting their rules and parameters. Toward this end, it is desirable to be able to leverage the power of other modeling methods, particularly those of other families: validation and analytical models.

The Communicating X-Machine (CXM) formalism has been used extensively for the validation of distributed software systems. Stamatopoulou et al. \[48\] have extended this model with a linkage to Population P Systems (PPS), a technique used for modeling biological systems of cells (developed as an extension to membrane computing’s P Systems). CXMs allow analysis of individual agents, while PPSes enable reasoning about aggregate behaviors and changes in the entity makeup of the entire systems.

Based on an analysis made via descriptive models, a validation model can be constructed. Similar to Sudeikat and Renz’s approach for transforming CLDs into an $\pi$-calculus model, we are investigating a related transformation from design patterns and CLDs to a CXM-PPS model.

- **Analytic Approaches**: Analytic approaches typically take the form of a mathematical model of the system that can either be directly analyzed (and hence optimized) \[59\] or statistically simulated.

A relevant approach using this latter technique is \[55\], which uses state partitioning with a Monte Carlo simulation. Also of particular interest, is the technique used by Renz and Sudeikat \[51\] to derive a system of differential equations capturing the state transitions in a system (using a method described by Cardelli \[5\]), starting from either a CLD or from a $\pi$-calculus model (via intermediate reaction equations). This approach allows the create of a simpler (and analytically tractable) continuous model that still reflects important properties of the system under design or consideration.

**Deliverable**: Using the selected array of approaches, we will analyze Myconet, a system with which we have extensive experience. Our previous work with Myconet will provide a basis from which we can evaluated the benefits and limitations of each modeling approach, and identify points in each where extension will be needed to enable inter-model transitions. The use of Myconet as a subject of analysis will also allow us to develop our evaluation strategy for the efficacy of individual modeling approaches: specifically, we will begin to refine the quantitative claims we can make about a system based on a collection of models. For a brief consideration of how these models might inform the design of a self-organization software system, refer to \[4.2\] below.
Stage 2: Working from our results from Stages 1, we will develop a preliminary integration of the selected models, resulting in a collection of transformations for moving between different models and a process for applying these.

The key continuing challenge in the design and implementation of self-organizing systems is understanding the global emergent behaviors and properties that result from rules followed by lower-level components. The design pattern approach is a useful stepping stone in providing leverage in this area: each pattern represents a well-understood piece of dynamics with its own micro-/macro-level linkages; thus, focus can be shifted to the higher-level dynamics that result from the composition of the identified lower-level mechanisms, and the effects of adjusting their rules and parameters. Since in systems with emergent behavior simple changes can result in large modification of the overall dynamics, the modularization of the system also improves the level of support for the maintenance and evolution of the system over time, as changes can be more easily targeted to contained areas of functionality.

**Deliverable:** By extending the design pattern notation to enable easy creation of models from the other families, we expect to be able to make headway with one of the significant open problems with self-organization design patterns: the composition problem. While a particular patterns of dynamics may have been studied extensively, the properties of self-organizing systems mean that when several patterns are used simultaneously, the resulting behaviors may not be easily predictable. Furthermore, such prediction is not readily tractable to direct analysis, so a toolkit of modeling approaches and a methodology for rapidly developing simulations of the dynamics of such systems is needed to support design and engineering.

Stage 3: The extended models and preliminary transitions developed in Stage 2 will be applied to the core Myconet platform, and also will be used to model the layered applications that have been developed on top of it. In particular, we will examine how these extensions can be expressed as specializations or extensions to the model of the core system.

This stage will result in continued refinement of aggregated modeling toolkit and collection of transformations, and a developed collection of the needed extensions identified in the previous stage. Once the collection of transitions has been defined, these can be used as the core of a unified methodology for the design and analysis of self-organizing software systems.

**Deliverable:** We will release Myconet and the collection of models, along with the integrated modeling toolkit and preliminary methodology for using the collection of transformations. This research will result in insights into which of the large number of models proposed in the research literature are most useful and beneficial for designing and analyzing self-organizing complex software systems in a unified way, as well as an integrated approach to applying those multiple modeling techniques.

This research will also provide a demonstration of a principled approach to the design of self-organizing applications using the Myconet platform, which will be instrumental in enabling its wider adoption. This unified modeling of Myconet may also serve as a blueprint or template of how the design and analysis of self-organizing applications in general can be addressed.
4.2 Example Scenario

We now briefly discussing a scenario applying these models to HITAP, the extension to Myconet that was discussed in 3.2.2. HITAP’s dynamics are implemented using a relatively self-contained set of self-organizing mechanisms that operate on top of the more complex Myconet dynamics, but these mechanisms are not completely isolated. They both affect the behavior of the the underlying overlay and are affected by its dynamics in return.

A unified set of modeling techniques could have facilitated the initial design and subsequent tuning of the desired self-organizing dynamics. The original engineering was performed by drawing on our experience developing similar systems, and the rules and parameters were then tuned through trial-and-error to function well within the desired operational conditions. Our goal is to achieve a more principled procedure by which we can use a collection of of models and transitions such as those discussed in above.

HITAP’s initial design was inspired by a natural metaphor, that of chemical signalling in biological systems. By approaching such a problem at design time with a collection of self-organization design patterns, we can identify the use of the the Gradient pattern, which uses Spreading, Aggregation, and Evaporation, as component sub-patterns.

This breakdown then led to an analysis via causal loop diagrams to examine the interrelations of the individual mechanisms and the rest of the Myconet system in which it is hosted (see the next chapter for this detailed analysis). In HITAP, targeted attacks (and consequent failures of high-degree superpeers) results in an increase in alert hormone concentration in the system (via Spreading and Aggregation, which, in turn results in an increased rate of peers switching into the Bulwark state. Similarly, peers switching into Bulwark cause their neighbors to generate yet more hormone, driving the alert hormone concentration higher: a reinforcing loop. Countering this is a balancing loop driven by Evaporation: evaporation reduces the level of alert hormone, resulting in peers switching back to normal operation. These two loops capture the primary dynamics of the self-organized mechanism that will require tuning, and allows an initial identification of parameters. (These loops are depicted in block E of Figure 5.1 in the next chapter.)

With these identified patterns and a collection of associated transformations, the next step would be to map the causal flows to validation and analytical models. Specifically, the pattern definitions can include modules that can be can combined in order to quickly construct a model of the planned system. Communicating X-Machines could be used to represent the individual agent behaviors, and the CXMs can then be composed and translated into a Population P System that reflects properties of the structure depicted in the CLD.

Similarly, a translation to an analytical model such as a system of differential equations (as in [55]) focuses on factors such as determining appropriate thresholds and coefficients for rules and parameters, allowing faster convergence on a design that fits the needs of the desired scenario. The use of models from several families allows the exploration of system behavior in a range of scenarios (such as different network conditions) without requiring the development of a full implementation.

Such a collection of models and transitions will enhance the engineering process for self-organizing software systems and shorten the iteration cycles over which the system dynamics are refined.

5. Preliminary Results

In support of the research program discussed in the previous chapter, we have begun to analyze Myconet and its Mycoload and HITAP extensions using the initial collection of models that have been identified as the most promising. We now present our initial results in support of this work, which we have currently been actively investigating.

Most research efforts in the literature thus far have been focused on the identification and extraction of self-organization design patterns, typically either focusing on a single pattern that characterizes the primary dynamic of a particular system or gathering together patterns proposed in other works. This section addresses the next stage of this problem: identifying patterns within an existing, complex distributed system that relies on multiple self-organizing mechanisms, and using those patterns as the basis for a modularized redesign of the system. We see this as a significant step toward the development of a methodology for engineering
self-organizing software systems.

Using the catalog of self-organization design patterns in [18], we reverse engineered Myconet and its extensions. This is an important step in stage 1 of our research agenda, and will serve as a base for the developments planned for the next stages. This exercise also resulted in the isolation of two new self-organizing mechanisms that are integral to the overall behavior of Myconet, Mycoload, and HITAP: (1) the Specialization pattern (as described in 5.1.1), which captures a mechanism that is present in a number of other self-organized systems and solves the recurring problem of self-selected differentiation of roles and responsibilities among (possibly heterogeneous) system elements or agents; and (2) the Collective Sort mechanism (described in 5.1.2) that enables collection of entities into similar-type groupings within the context of a collection of system elements. This sort of clustering is also a recurring problem in self-organizing systems, where elements often need to be assorted into homogeneous groupings according to some property of the entities or requirement of the system.

The case study demonstrates the effectiveness of a well-organized catalog of design patterns [18] in modeling the design of an existing self-organizing software system. Many of the patterns of self-organization identified in Mycoload are at the basic level, and as such are ideal candidates for reuse or external implementation via a self-organizing middleware.

Beyond demonstrating the usefulness of a catalog of self-organization design patterns, another benefit of the design analysis was to fulfill the first two phases of stage 1 of the research plan discussed in this proposal: analyzing Myconet and its extensions using layered design patterns and causal loop diagrams. We have begun the next phase of stage 1 by developing models of this system using Communicating X-Machines and Population P Systems. The process of creating these models will lead to the creation of our first set of inter-model transformations.

5.1 Design Patterns for Self-Organization

In Chapter 4 we discuss initial work using a catalog of patterns for the modeling the dynamics of a specific self-organizing application (Mycoload) that uses a wide variety of mechanisms. This modeling effort has also proved useful in the highlighting and isolation of other reusable self-organizing mechanisms, and led to the identification of two new patterns: Specialization and Collective Sort. These patterns

5.1.1 Specialization Pattern

Specialization is a mechanism widely used in complex systems for achieving improved efficiency by exploiting the natural heterogeneity of the entities taking part in the system. Through Specialization, each individual entity is assigned a specific role depending on its capabilities and contextual local information. This assignment can be be considered as “closed” specialization when the set of possible roles is known a priori, or “open” specialization otherwise. A useful survey of specialization in self-organizing systems can be found in [36].

According to the Specialization patterns, system entities will change the rules under which they operate, depending on features or properties of the entity itself, or contextual information from its environment and neighbors. For example, a computer with high amounts of memory available could store information on behalf of nodes with low memory; a computer with sensors could provide sensed information to other computers; a network node with plenty of bandwidth connections could be elected to act as a router for traffic transmitted by other nodes, and so on. The assumption of specialized roles may be influenced, or may need to be further modulated, to adapt to changing system conditions and requirements in the system.

**Name:** Specialization

**Aliases:** None to our knowledge.

**Problem:** Global optimization of system efficiency by increasing or decreasing the contributions of individual entities or by otherwise changing the rules under which those entities operate.

**Solution:** Individual entities are assigned a specific role or set of behavioral rules depending on their capabilities and contextual local information. Specialization optimizes entities’ contributions in order to increase the overall performance of the system.
Inspiration: The specialization process appears in many macro- and micro-level systems. Some examples are the specialization of cells in a human body or the specialization of individual humans to fill particular roles in society.

Forces: Depending on how the contextual information used for making decisions regarding specialization is acquired and which patterns are used in transferring and maintaining this information, different trade-offs can appear. The most common patterns are Spreading and Aggregation (see the forces discussed in their pattern descriptions [18] for more details.) In general, though, the information used by Specialization can come from any other self-organizing mechanisms or a combination of those mechanisms, and their dynamics will influence and possibly be mutually influence the resulting specializations.

Entities: The entities participating in the Specialization pattern are: (1) software agents that modify their behavior depending on their capabilities (or the capabilities of their hosts) and environmental information (e.g. external requirements); (2) hosts that provide sensors, memory, communication capability, computational power, etc. to the software agents; and finally (3) Environment, all that is external to the hosts (e.g. the space where host are located, external requirements that are injected in the system, etc….)

Dynamics: Agents retrieve contextual information from their own knowledge and from their neighbor agents, or from the environment by using sensors or querying an externally implemented environmental model. To describe the dynamics we use the same notation as in [18], where information contained in the system is modelled as a tuple \( (L, C) \), where \( L \) is the location where the information is stored (possibly within an agent or maintained by an external middleware), and \( C \) is its current content—e.g. in the form of a list with one or more arguments of different types, such as numbers, strings or structured data, according to the application-specific information content. Transition rules resemble chemical reactions between patterns of tuples, where (i) the left-hand side (reagents) specifies which tuples are involved in the transition rule: they will be removed as an effect of the rule execution; (ii) the right-hand side (products) specifies which tuples are accordingly to be inserted back in the specified locations: they might be new tuples, transformation of one or more reagents or even unchanged reagents; and (iii) rate \( r \) is a rate, indicating the speed/frequency at which the rule is to be fired (that is, its scheduling policy).

\[
\text{state\_evolution} :: (L, [c\text{Inf}, \text{State}, C]) \xrightarrow{\pi} (L, [c\text{Inf}, \text{State}', C])
\]

where \( \text{State}' = \pi(c\text{Inf}, \text{State}, C) \)

In the above rule, \( c\text{Inf} \) is the contextual information accessible to the agent. \( \text{State} \) is its previous role before the specialization occurs (i.e., the set of rules under which it operates), \( \text{State}' \) is the new role (and consequent set of rules) adopted as a result of the specialization process, and \( \pi \) is a function that produces this new state from the given contextual information, current agent state, and any local information.

Environment: The hosts must have different features or the system must display other heterogeneities (for example, in the distribution of agents in different locations) that allow Specialization to assign an appropriate role based on those features and the contextual information.

Implementation: We have identified two different implementations: (1) An agent decides to change its role in the system by taking into account the capabilities of the local environment where it resides and acquired contextual information (e.g., the system’s requirements); and (2) An agent is positioned to determine that one of its neighbors should adopt a new role. An example of the second case is when one node is providing services to other nodes in the system but it is reaching the maximum number of clients; in such a case the node can replicate the information served to a new node (selected according to some suitability measure from among other nodes in its neighborhood) and target it to assume the role of an additional service provider.

Two types of rules can be used to drive the specialization of agents, determining which role an individual adopts depending on its capabilities and context: fixed rules, and adaptive rules. Fixed rules are defined by developers at design time, agents switch among behaviors from a static set. Adaptive rules may be changed by the agents during run time in order to contribute to the optimization of the global system behaviour. Evolutionary approaches have been used in the field of autonomic computing to establish sets of norms, policies or rules that drive the system to the desired emergent behaviour, even when environmental changes occur. A possible implementation was introduced in [43], which uses a distributed genetic algorithm. In that approach, each agent participating in the system performs local evaluations and adjustments that are then shared with other agents using spreading mechanisms.

Known Uses: Specialization has been used by a large number of self-organizing applications. Examples include: (1) Overlay networks where some nodes decide to become routers based on their available resources and their connectivity with other nearby nodes [27] (2) Aiming to localize diffuse event sources in dynamic environments using large scale wireless sensor networks, agents change their roles in order to locate and track diffuse event sources [19]. (3) To balance the load among nodes with different services, Mycoload [54], builds a superpeer topology where more powerful nodes adopt several different, specialized roles in the creation and maintenance of the overlay. (4) [35] describes a robust
Collective sort is a clustering mechanism that enables segregation or relocation of entities into similar-type groupings within the context of a collection of system elements according to some property of the entities or requirement of the system \cite{56,50}. Self-organizing algorithms for collective sorting have been developed based on observations of biological phenomena, particularly the processes of brood sorting and cemetery formation by social insects \cite{59,44}.

The mechanism discussed in this pattern is a generalization of the biologically inspired collective sorting proposed as a design pattern for tuple spaces by Gardelli \emph{et al.} \cite{23,22} and analyzed as an environmental coordination mechanism by Sudeikat and Renz \cite{50}. The usual formulation of the collective sort algorithm assumes a case where active agents relocate inactive data items; the Collective Sort pattern presented in this paper extends this to include cases where the agents themselves may be the entities to be grouped, or where different environmental abstractions are being used. These variants are discussed in the Implementation section of the pattern description, below.

\begin{itemize}
  \item **Name:** Collective Sort
  \item **Aliases:** Brood Sorting, Cemetery Formation, Collective Clustering
  \item **Problem:** A system contains a number of scattered data or entities that need to be brought into relative proximity with other similar data or entities.
  \item **Solution:** Individual agents move through an environment, encountering data items as they travel. By picking up and dropping these items based on local heuristics, elements with similar properties are progressively gathered into homogeneous groups or clusters.
  \item **Inspiration:** Brood sorting and cemetery formation by social insects \cite{59,44}
  \item **Forces:** This pattern starts from a disordered arrangement of entities within an environment and progressively reduces that disorder. Thus, it is affected by entity distribution in the environment, and other other forces that act to change the location of these entities. The function used by entities to evaluate local density of entities (as well as an entity’s range of perception) will affect the outcome of the sort. In particular, a small range may induce the formation of multiple small collections. The choice of rules and probabilities for the picking up and dropping of entities may also influence the speed of convergence or the resulting topological distribution.
  \item **Entities:** The entities associated with the Collective Sort pattern are: (1) data items that have an associated property for which similarity can be assessed; and (2) active agents that are able to examine and relocate data items of type (1). Note that, depending on the implementation, (1) and (2) may be the same entities. For example, the active agents may themselves possess the property that is the subject of sorting, and hence the emergent order will be expressed by the arrangement of the agents themselves.
  \item **Dynamics:** This pattern relies on three rules which, together, tend to progressively relocate similar elements into similar vicinities: a \textbf{Movement Rule} that relocates agents within a set of candidate locations, a \textbf{Pick-Up Rule} that connects an agent to an element so it can be moved, and a \textbf{Drop Rule} that leaves a held element at a current location. These rules are followed by the active agents. The Movement rule is frequently implemented as random exploration, but could also take advantage of available contextual information if appropriate. The Pick-Up and Drop rules select entities to be clustered when they are encountered, tending to remove elements from areas of high diversity and deposit them in areas of low diversity. Two approaches to applying these general rules are discussed in the Implementation section, below.
  \item **Environment:** The environment provides the context within which the proximity of data items is interpreted. Thus, entities must have a concept of location, be able to change location, and be able to detect other nearby entities within that environment; they must also have a means of evaluating the similarity of data items thus detected.
\end{itemize}
Implementation: Different implementations are possible. We here discuss two scenarios: a distributed tuple space and a peer-to-peer network.

In a tuple space implementation active agents typically move from tuple space to tuple space, carrying tuples with them. An example of rules implementing a biologically inspired variant of Collective Sor from this case (derived from [6]) is:

1. Movement Rule: Agents explore the environment, encountering data items as they move. The choice of movement strategy is frequently random, but may also be informed by other information, such as using a Chemotaxis strategy. An example of a random exploration rule for an agent is:

   $\text{random} \rightarrow \langle L, C \rangle \mapsto \langle L', C \rangle$ where, $L' = \text{random}(\text{NEIGHBORHOOD}(L))$

2. Pick-Up Rule: Agents “transport” data items from place to place, tending to move them from areas with lesser concentration to areas with greater concentration of items of the correct type. This rule may rely on direct observation of data items in a vicinity (as with observation of neighbors in Flocking); it may rely on Aggregation to estimate the local density of data items of a particular type; or the agent may make its own estimate by maintaining a memory of recently encountered data items. A general way to express how pick-up occurs is through a probabilistic function of the density of data items. For example, the probability of picking up an encountered data element of type $t$ at location $L$ might be:

   $$P_{\text{pick-up}} = \left( \frac{k_{\text{pick-up}}}{d(L)} + d'(L) \right)^2$$

   where $d'$ is the estimated density of data items of type $t$ at location $L$, and $k_{\text{pick-up}}$ is a constant; the associated rule when the agent encounters element $e'$ of type $t$ in the local environment $E$ would be executed with probability $P_{\text{pick-up}}$.

   $\text{pick-up} :: \langle \langle L, C \rangle, E \rangle \xrightarrow{\text{pick-up}} \langle \langle \langle L, C \cup \{e'\} \rangle, E - \{e'\} \rangle \rangle$

3. Drop Rule: Similar to the Pick-Up Rule, agents will drop items when they estimate that the local vicinity is propitious. The probability of dropping a held $e'$ element of type $t$ at location $L$ with environment $E$ might be:

   $$P_{\text{drop}} = \left( \frac{d'(L)}{d + d'(L)} \right)^2$$

   and the associated drop rule, executed by the agent with $P_{\text{drop}}$, would be:

   $\text{drop} :: \langle \langle L, C \rangle, E \rangle \xrightarrow{\text{drop}} \langle \langle L, C - \{e'\} \rangle, E \cup \{e'\} \rangle$

In a more general view of Collective Sor, an abstract notion of grouping can be used to apply the same strategy in a scenario where the environment is defined by a pattern of neighbor relationships between nodes (composing a graph, as in a P2P network), and where the nodes themselves are labelled with some property (e.g., a node type) upon which clustering should be performed. In this dynamic graph scenario, “movement” is considered to be selecting a candidate location for growing a new neighbor relationship, “picking up” is adding a new neighbor, and “dropping” is severing an existing neighbor relationship.

1. Movement Rule: Agents explore randomly by selecting a potential new neighbor from a set of possible candidates. In many P2P networks, and in Mycoload, this set is maintained by a separate mechanism that implements the Gossip pattern, and thus provides each node with fresh samples of candidate non-neighbor nodes. For a node $V$ with neighbor set $N(V)$, a new possible neighbor $C$ is selected:

   $\text{graph} \rightarrow \langle V, N(V) \rangle \mapsto \langle V, N(V), E \rangle$ where, $C = \text{random}(\text{CANDIDATES}(V))$ and $C \notin N(V)$

   Note that this may also result in $V$ finding a cluster of its own type (if the new neighbor $W$ is of the same type) if it was not already in one, or finding a connecting path for two disconnected same-type sub-clusters.

   2. Pick-Up Rule: Once the movement rule has selected a new possible neighbor $C$, the agent may add it as a new neighbor to itself

   $\text{graph} \rightarrow \langle V, N(V), C \rangle \xrightarrow{\text{pick-up}} \langle V, N(V) \cup \{C\} \rangle$

In Mycoload, for example, the pick-up rule will be executed if (a) $V$ does not currently have a neighbor that is its same type, (b) if $V$ has under a certain number of different-type neighbors, or (c) execute anyway with a small probability to prevent the system from settling into a local optimum. Thus, agents will wander until they find a cluster of the same type (whether by encountering it by moving or dropping in place by another agent), but will also try to keep connections to
neighbors of other types in order to help other nodes move toward an appropriate cluster. The pick-up rate thus declines as the nodes converge toward clusters.

3. Drop Rule: Dropping for a node V is performed by randomly selecting a neighbor W ∈ N(V) (where the type of W with neighbor set N(W) is different from the type of V) and, if V also has a neighbor U ∈ N(V) with the same type as W, by transferring W to become a neighbor of U:

\[ \text{graph_drop} :: (V, N(V)), (W, N(W)), (U, N(U)) \xrightarrow{\text{drop}} (V, N(V) - W), (W, N(W) \cup \{U\}), (U, N(U) \cup \{W\}) \]

A more detailed discussion of the role of Collective Sort within the Mycoload application is provided in Section 5.2 below.

**Known Uses:** Collective sort and brood sorting-inspired approaches to distributed self-organizing systems have been applied to several problems areas: (1) Collecting similar tuples from a distributed tuple system into a single tuple space [23]. Of interest is Casadei et al.’s approach using “noise” tuples to implement a simulated annealing-type approach to avoiding local optima [7]. (2) Collective sorting as a coordination mechanism for swarms of self-organized robots, proposed as early as 1991 by Deneubourg et al. [14] and extended by later researchers [56]. (3) Storage and retrieval of Semantic Web documents. Presenting such a scenario, Muhleisen et al. [34] discuss in particular the role of similarity metric selection in collective sorting. (4) Intrusion detection. Sudeikat and Renz [50] identify brood sorting as suitable for providing a portion of the self-organizing dynamics for a stigmergy IDS. (5) Clustering of same-type nodes in peer-to-peer networks. Mycoload [33] uses a collective sort approach to build clusters of peers offering the same service types; the specific role of the collective sort mechanisms is discussed in this paper.

**Consequences:** Collective sort enables clustering of data or other entities into groups of similar type. The resulting order may increase efficiency of operations on this data.

**Related Patterns:** **Aggregation** is often used to estimate local density of data items. **Spreading** and **Gradient** may be used to disseminate information about the density of particular kinds of data, and **Chemotaxis** may be used to guide agent movement. **Specialization** may be used to select specific items among the ones in the resulting groupings for particular roles.

### 5.2 Case Study: Modeling HITAP

We now present our analysis of Myconet and the Mycoload and HITAP extensions using design patterns (including the **Specialization** and **Collective Sort** mechanisms discussed above), and the causal loop diagrams derived from this reverse engineering exercise. This work has already enabled a refactoring of the existing Myconet codebase, allowing us to isolate individual mechanisms that had previously been entangled in the existing implementation.

The primary goal of Mycoload is to maximize throughput by balancing load queues for multiple services across many peers in a decentralized service network with heterogeneous capabilities. We decomposed this goal into a set of primary features provided by the system. First, in order to construct an efficient overlay network, Mycoload relies on Myconet’s features, through which (A) peers discover appropriate candidates to become neighbors when entering the system and while maintaining the overlay; and (B) the overlay exploits the heterogeneity of the system by promoting higher-capacity peers to specialized roles with higher responsibilities in the maintenance of the overlay. The Mycoload extensions add additional features to the overlay; specifically, these extensions (C) collect peers offering the same service into connected subgroups, so that they can interchange jobs with other peers in the same subgroup, and (D) shift jobs between peers such that peers with higher capacity are given proportionally more work, and thus performs load balancing of job requests. Finally, the HITAP extensions (E) protect the overlay in the face of targeted attacks.

The breakdown of Myconet, Mycoload, and HITAP according to the five major features specified above is shown in Figure 5.1, which depicts representation of the design as an annotated causal loop diagram (CLD). CLDs are often used to describe and reason about Complex Adaptive Systems, and have been used by Sudeikat and Renz [32] for analyzing self-organizing coordination mechanisms. We have adopted a CLD view of our design because, in comparison with other design modeling facilities—such as UML structural diagrams (e.g., component diagrams) or interaction diagrams (e.g., sequence diagrams)—CLDs make explicit the feedback loops which exist in self-organized systems, and which are essential to their functionality.

Our reverse-engineering exercise has shown how multiple feedback loops are present in the layered Myconet applications, and how they interact with one another. Some loops correspond one-to-one to self-
organization design patterns that we have been able recognize and isolate; in other cases, patterns—especially those from the basic layer of our catalog—are best associated with transitions (arrows) within a loop; in yet other cases, certain transitions can be associated to one of the transition rules that we have defined in the body of a pattern specification (as in block (C) of the figure). We highlighted with different textual or graphic annotations each of those cases in our diagram: the Myconet features discussed below are highlighted by colored, rounded rectangles around the corresponding portions of the diagram. High-level patterns are shown as squared rectangles labelled with a monospace font. Grey rectangles oriented vertically and associated with transitions indicate basic patterns, and larger shaded rectangles cover a range of states associated with a higher-level pattern. Vertically oriented monospace text without a grey box indicates a rule that is part of a pattern.

The "by-feature" decomposition of the design proposed in Figure 5.1 has allowed us to see where each loop (and therefore each design pattern) operates in the system, and how it influences other loops and other features. In the remainder of this Section, we discuss each area of this composed CLD diagram.
(A) Discovery of candidate neighbor peers

Peers need to be able to discover other peers when entering the system in order to create new, appropriate neighbor relationships for building and maintaining the overlay. Ideally, a system with perfect knowledge would be able to select a suitable peer from anywhere in the overlay. In practice, given the decentralized and dynamic nature of large-scale peer-to-peer overlays, peers must achieve this through a mechanism that works locally and maintains a limited set of data.

One well-studied mechanism for that is the Gossip pattern, which allows peers to share knowledge about other active peers, outside of the context of established neighbor relationships. With each interaction, pairs of peers contribute to a joint agreement on a set of nodes that are likely to be fresh (i.e., likely live and participating in the overlay); this information is then spread to other nodes through subsequent exchanges. The Gossip mechanism is composed of two sub-patterns, Spreading and Aggregation. Both mechanisms, in this case, operate on a list of known peers that is maintained at each local node. Each peer periodically selects a known peer and sends a copy of its entire list (Spreading). Upon receiving such a message, the recipient combines that list with its own (Aggregation), along with an entry for the sending peer timestamped with the current time. In order to prevent the lists from growing indefinitely (and to eventually remove peers that are no longer participating in the gossip exchanges), peers are removed from the list via an Evaporation mechanism. Entries with the oldest time are progressively discarded, keeping the list at a fixed length.

In Mycoload, Gossip as well as all of its component patterns are implemented by the Newscast protocol [26], which Mycoload adopts as an off-the-shelf component in order to fulfill feature (A).

(B) Peer Exploitation

Mycoload is designed to exploit communities of peers with highly heterogeneous capabilities and resources (that is, where some peers are much more powerful and have a much higher capacity than others). Heterogeneity brings about an opportunity for improved performance, with a limited number of peers with more capabilities tasked to perform certain advanced functions; this is a basic functioning principle for all super-peer overlays, and represents a typical use case for the Specialization pattern.

In Mycoload, each peer considers its own status and the situation in its neighborhood (primarily, its number of neighbors relative to an ideal capacity, a measure based on its available resources, and its own capacity compared to that of its neighbors). Each peer uses this local information to determine whether to adopt a new role (as a superpeer in one of the hyphal stages described in Section 3.2.1), and switching the set of protocol rules it uses to manage neighbor relationships accordingly.

The information considered by a peer when making the decision on whether to adopt a new, specialized state may also include messages from other peers (in this case, superpeers with a broader view of the network); these are received “hints” that the peer is well-positioned to be promoted and serve as a new superpeer, or that it has now become redundant, and it can reduce inefficiency by demoting itself. A full description topology maintenance rules followed in each specialized state cannot appear here due to space limitations, but can be found in [54].

(C) Collect peers offering similar services into groups

In the service network scenario targeted by Mycoload, peers may offer different services, and jobs of a certain type may only be processed by peers offering a service of the same type. In order to exchange jobs and balance their respective loads, peers of similar type need to be able to find, group with, and collaborate with each other. To enable that, the Collective Sort pattern operates to adjust the topology of the overlay in such a way that neighborhoods of peers with the same service type will emerge. Peers within a neighborhood can exploit the heterogeneity their processing capabilities by engaging in the specialization pattern, so that each neighborhood of same-type peers coalesces and stabilizes around one or more hyphal peers of the same type.

The clustering dynamics of the Mycoload extension thus provide additional self-organized logic, which extends, with considerations of neighbor types, the basic topology dynamics that regulate the construction of the superpeer overlay, as per feature (B). That additional logic underlying the Mycoload implementation
of Collective Sort induces peers that have not managed to achieve a target number of same-type neighbors to explore their environment by querying neighboring superpeers for other same-type peers; the superpeers act as matchmakers, attempting to connect their own neighbors with same-type peers that are one link removed. Notice that any peers that are unable to find matches using these mechanisms also engage in random exploration, using the peer list obtained through Gossip as per Feature (A).

The specific rules used by Mycoload to implement its version of Collective Sort are the versions of graph_movement, graph_pick_up, and graph_drop discussed in Section 5.1.2.

(D) Load Balancing

Since all the responsibility for building, maintaining and adapting the overlay is handled by features (A), (B) and (C), the self-organized load balancing can rely on the properties of the resulting overlay. Load-balancing has thus simply the responsibility of equalizing levels of work among peers offering the same type of service, scaled according to each peer’s capacity.

This is a natural match for the Gradient pattern. The jobs in the queue of a peer are considered analogous to the quantity of some chemical marker, and the peer capacity equates to a volume in which the chemical is present. A peer can easily determine the relative concentration of the chemical marker relative to its neighbors, and, based on this information, can determine whether some number of jobs need to be transferred in order to achieve a balanced load. As a result, excess jobs diffuse through the system, with more jobs proportionally accumulating at higher-capacity peers.

Gradient is implemented once again through the basic patterns of Spreading and Aggregation. For the Spreading portion, peers have access to information about their direct neighbors via the overlay network, including their capacity and queue length. Peers periodically randomly select a neighbor with which to balance queues, and the peer with the higher relative concentration transfers the job to the other. Aggregation is performed when a peer receives jobs from a neighbor and they are merged into its own queue, ordering them such that each receives fair service based on its time of entry and other features.

(E) Self-Protection

As with features (C) and (D), the self-protection functions of HITAP are isolatable from the other components of Myconet. Indeed, the mechanism could be used with a completely different overlay topology management protocol. The self-protection approach taken uses a straightforward hormone signalling mechanism, with diffusion implemented via a Gradient mechanism. Upon failures, the generated alert signal is passed to neighbors via a component Spreading pattern implementation, whereupon Aggregation combines signals from multiple sources. Once a threshold is passed, a node switches into the self-protective bulwark state, triggering further releases of hormone in its neighbors and making a reinforcing feedback loop. The build-up of hormone levels is balanced by Evaporation, which reduces the concentration levels in the system and drives nodes back into normal operations.

6. Conclusion

The need for a set of tools allowing reusable design, validation, and analysis has been highlighted by our previous experiences building self-organizing systems. These concerns are all important modeling dimensions, and the next steps beyond the current state of the art will require connecting models focusing on distinct concerns in order to achieve a comprehensive, insightful representation of a self-organizing computing system across those three dimensions. A major goal of this research work is to identify and select the best available modeling techniques from each of these families, and integrate them to better support the creation of new systems, as well as the analysis of existing ones.

Myconet has proven to be flexible, and we have started to use it as a platform for other self-organizing applications in large-scale distributed systems. Those works have demonstrated the applicability of Myconet to a variety of self-organizing scenarios and applications. However, to fully leverage some of the Myconet
properties, application-level mechanisms may need to interact with the lower-layer Myconet topology management protocol, and may induce alterations to the basic overlay behavior. The most difficult task when developing a new extension is often to fully understand the impact of the interactions of composed self-organizing mechanisms, and to account properly for those repercussions from the beginning of the design and implementation activities.

We have examined a number of models and tools from the literature. The goal of this research is to select a small set of synergetic techniques, and develop a comprehensive, integrated suite of modeling approaches, which cover the design, validation and analysis dimensions. Armed with that suite, we intend to model Myconet and its extensions (either already developed ones or new and upcoming ones). That way, we will experiment with and validate its modeling capabilities, and demonstrate how it can be used to acquire early on an accurate insight on how to either design new self-organized software, or approach the maintenance of existing self-organizing features, in the face of novel or changed requirements.

This research will result in a collection of modeling techniques and transformations for moving between them that can be used for the engineering and analysis of self-organizing software systems, as well as in insights into which of the large number of models proposed in the research literature are most useful, as well as an integrated approach to applying those multiple modeling techniques. It will also provide a demonstration of a principled approach to the design of self-organizing applications using the Myconet platform, which will be instrumental in enabling its wider adoption. This unified modeling of Myconet may also serve as a blueprint or template of how the design and analysis of self-organizing applications in general can be addressed.
Bibliography


