A Case for Design Methodology Research in Self-* Distributed Systems

*Indranil Gupta, Steven Ko, Nathanael Thompson, Mahvesh Nagda,
Chris Devaraj, Ramsés Morales, Jay A. Patel

Dept. of Computer Science
University of Illinois at Urbana-Champaign, Urbana IL 61801
{indy,sko,nathomps,nagda,devaraj,rvmorale,jaypatel}@cs.uiuc.edu

Abstract. We argue that "design methodology research" for self-* distributed systems needs to be recognized and enriched. Methodologies encourage systematic design of distributed protocols. They augment the creative activity of innovation, rather than stifle it. They enable easy design of, and automatic code generation for, distributed systems with predictable properties. Through a taxonomy, we show that methodology research is growing slowly but steadily. As a case study, we present and discuss a new methodology that concretely captures the design of a large class of peer-to-peer distributed hash tables (p2p DHTs) and DHT-based applications. We use this to show some advantages of methodology research, such as effective exploration of the design space for protocols. We also summarize some of our ongoing work in the direction of developing methodologies for distributed protocols.

1 Introduction

Today, designing new protocols for self-* distributed systems such as peer-to-peer (p2p) systems, autonomic distributed systems, Grid applications, etc., is an extremely challenging task. Consider a researcher who is asked to design a distributed protocol for a specific p2p application with certain properties. The only resources available to the designer are 1. her basic distributed systems knowledge, 2. prior research literature, and 3. designer's experiences.

This is almost a "seat of pants" approach to protocol design. It has resulted in long research project timelines, as well as long lag times to production and deployment (anecdotes suggest 5 to 15 years). Resultant designs may also be complex, with massive code line counts and inefficiencies when pieces are put together [5, 16].

These shortcomings can be addressed for future systems by populating and enriching a fourth resource for the protocol designer – Protocol Design Methodologies. A protocol design methodology can be loosely characterized as *an organized, documented set of building blocks, rules and/or guidelines for design of a*

---

class of distributed protocols. It is possibly amenable to automated code generation.

Given a distributed computing problem then, a collection of methodologies can be brought to bear, for either innovating novel protocols, or for composing existing protocols. This would create a range of simple and efficient solutions to the problem at hand, thus offering several choices for selecting the most appropriate design. In general, the application of methodologies results in a more *systematic* approach to design. They augment the creative activity of innovation, rather than stifle it. In the long run, short design times, and compact and efficient protocol designs, are possible side-effects.

Many disciplines have already used methodologies to systematize and mature the creative design process. Since the early days of the Internet, TCP/IP layered architectures have helped decentralize design responsibility for infrastructure and applications among different research communities. Hardware design uses automated synthesis [2], and software engineering uses Design Patterns and Model-driven architectures [8]. Similar maturity for distributed systems research requires creation of a critical mass of methodologies, and this can be achieved through *Design Methodology Research*. As we show in this paper, the discovery of design methodologies can go hand in hand with protocol design itself.

We make our case for methodology research in distributed systems by first recognizing that there are already several methodologies for protocol classes. We present a new taxonomy for classifying methodologies. Then, as a case study, we present a new *automatable* methodology that retroactively fits a large class of p2p DHTs and DHT-based applications. We show how this enables exploration of design space of DHT-based applications, and present experimental results to justify the benefits.

It will be evident from this paper that the elements of methodology research are more challenging than, and different from, those of hardcore software engineering. Developing a design methodology requires in-depth understanding of the protocols being designed, or the desired properties from the protocol, or both. A design methodology inherently captures the designer's frame of mind and philosophy, a goal that software engineering does not aspire to. The only connection with software engineering lies in the development of automated toolkits that enable a designer to generate working code for a new protocol. Although these toolkits clearly have a very different aim than existing Integrated Development Environments (IDEs), it is possible that the two might be consolidated in the future.

Previous Work: We briefly summarize some of the design methodologies that have emerged for distributed systems, not necessarily restricting discussion to self-* systems. It should be noted that some of these methodologies have had considerable impact, while others are less popular, and for many more, it is too early for the jury to be called out. For example OSI-like architectures are less popular than the omnipresent TCP/IP design methodology. On the other hand, the new methodology for DHTs presented in this paper is too young to be judged.

The set of existing design methodologies includes protocol families for survivable storage systems [22], Internet routing protocols [23], peer-to-peer systems [12,15], extensible router and OS design (e.g., [13]), I/O automata [21], etc.

Strategy design patterns [9] can be used to construct object-oriented code for a large class of deterministically reliable distributed protocols such as consensus. Other tools in this class include ASX and Conduits+. These works have been labeled as *microprotocols*, and include x-kernel based microprotocols and run-time composable systems [20]. Stack-oriented distributed systems such as Horus [19], and composable web services [17] are some other methodologies. However, all of these systems have composition rules that are based on function calls – we have discovered some methodologies that use more complex notions of composition (see Section 3). The above list is by no means comprehensive, and is only a snapshot of the slowly growing body of methodology research.

A new methodology we present captures some popular DHTs and p2p applications. DHTs include Pastry, Chord, Tapestry, Kademlia, Kelips, etc., and applications include CFS, PAST, Bayeux, Squirrel, etc. [1].

Section 2 presents a taxonomy of methodologies. Section 3 presents the p2p methodology, and Section 4 discusses experimental data. Section 5 briefly details our current work. We summarize in Section 6.

2 Taxonomy of Methodologies

In order to motivate an understanding of the features of methodologies, we present a new taxonomy of classification for them.

Formal vs. Informal: We call a methodology that is specified using precise rules or a stringent framework as a *formal* methodology, otherwise we say that it is *informal*. These "rules" could either be mathematical/logical notation, or the grammar of a high level programming language. Respective examples are the probabilistic I/O automata [21], and the methodology of [11] that takes as input a set of differential equations (satisfying certain conditions), and generates code for a distributed protocol that is equivalent. Previous methodologies for DHT design [12,15] have been informal.

Due to their rigor (either through a formal framework or a compiler), formal methodologies have the capability to create protocols with predictable or provable properties, and also to generate protocol code automatically. For example, the distributed protocols generated from differential equations in [11] are provably equivalent to the original differential equation, and can be generated by a toolkit called DiffGen [11]. On the other hand, informal methodologies are less rigorous and more flexible, but can have multiple possible interpretations. An informal methodology could be converted into a formal one through implementation of a specific interpretation. For example, an informal probabilistic protocol composable methodology can be instantiated through a high level language called Proactive Protocol Composition Language (PPCL) [10,18], thus making it formal.

METHODOLOGY TYPES	Innovative	Composable
Formal	Protocols from Differential Equations [11]; Bluespec for hardware synthesis [2].	TCP/IP layered architecture; Extensible router and OS designs (e.g., Click [13], SPIN, x-kernel [3, 20]); Routing [23]; Probabilistic I/O automata [21]; Strategy Design Patterns [9]; Stacked architectures (e.g., Horus [19]).
Informal	Design Patterns [8].	DHT design methodologies [12, 15]; Protocol family for survivable storage [22]; Probabilistic protocols [18].

Table 1. How Existing Methodologies Fit into the Proposed Taxonomy.

Innovative vs. Composable: Design methodologies must be capable of assisting in innovation of new protocols, as well as in the ability to reuse and adapt existing protocols. These are achieved respectively through innovative methodologies and composable methodologies. An innovative methodology describes how completely novel protocols can be created, e.g., [8, 11]. A composable methodology typically describes *building blocks* and *composition rules or guidelines*. Building blocks are either standalone protocols or strategies, and composition rules help combine the blocks to create new protocols with enhanced properties. For example, the informal methodology for DHT design in [12] uses four types of building blocks - overlay, membership, routing, and preprocessing. Strategy design patterns are another example of a composable methodology [9].

Table 1 summarizes the above discussion.

Discovery of Methodologies: Different approaches are possible for the *discovery* of these methodologies:

1. **Retroactive**: A methodology is discovered for an existing system or class of protocols. Ex: methodologies for routing [23] and probabilistic protocols [10].

2. **Progressive**: A methodology is invented that creates a novel class of protocols. Ex: the design of protocols from differential equations can generate new protocols for dynamic replication and majority voting [11].

3. **Auxiliary**: A methodology is discovered to assist and complement an existing methodology. Ex: protocol families for survivable storage architectures [22] combine several auxiliary methodologies for differing system models.

3   A New Concrete Methodology for P2P Applications

During the past few years, p2p researchers have designed over 25 different DHTs and DHT-based applications. Manku [15] and Iamnitchi et al [12] gave *informal*

design methodologies for DHTs. Here, we present the first *formal* composable methodology for a large class of DHTs and DHT-based applications. The DHTs covered include (but are not limited to) Chord, Pastry, Tapestry, Kademlia, Gnutella, CAN, and Kelips[1].
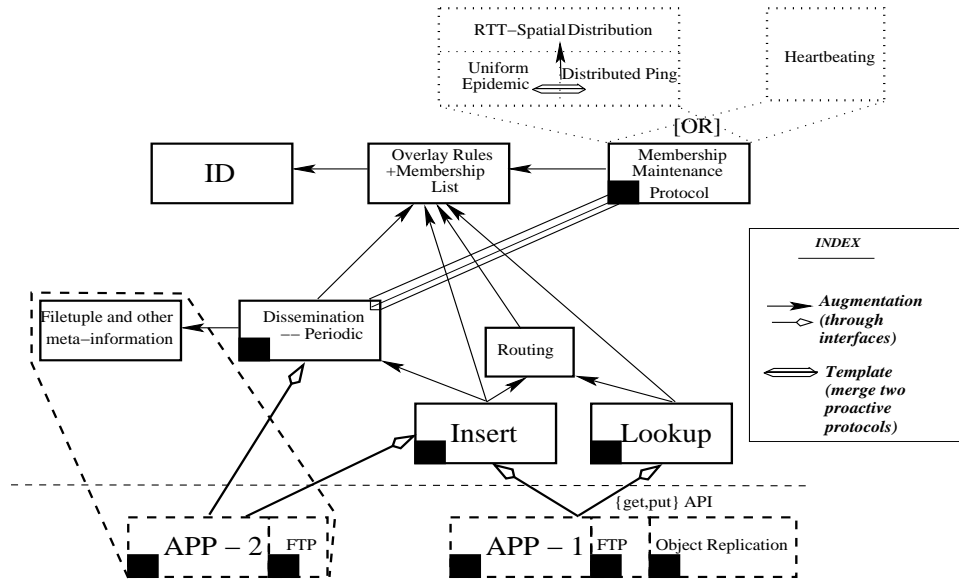
Fig. 1. A New Formal Composable Methodology for the design of a class of DHTs and DHT-based applications. *Covered DHTs include Chord, Pastry, Tapestry, CAN, Kademlia, Kelips. Covered DHT applications include Scribe, PAST, CFS, Squirrel (all App-1) and Kelips-based cooperative web caching (App-2). The methodology consists of standalone building blocks, composed by either (i) (augmentation) simple function calls or (ii) (template) by merging two periodic/proactive blocks together. Composition rules are shown in detail in Figure 2. Building blocks with dark shaded bottom-left corners represent functions that directly talk to their peer functions on other clients.* **This methodology is both retroactive (i.e., fits existing designs) and progressive (i.e., can be used for completely new designs).**

Figure 1 summarizes the methodology. This methodology can be used to create any of the system designs mentioned above. The figure shows several *standalone* building blocks, both at the DHT layer and for two models of DHT-based applications. The building blocks for DHTs include ID (assigns virtual id's to nodes), Overlay Rules and Membership List (encapsulates the overlay structure and maintains neighbors), membership maintenance protocol, dissemination protocol (for updating meta-information such as about files), routing, insert and lookup, and a block that stores meta-information. Two application

---

[1] The proof discussion of this statement is omitted.

models (App-1 and App-2) are shown, which we discuss shortly. Blocks with dark corners communicate over the network with corresponding block on peers.

These building blocks are composed by one of two rules - (i) augmentation (i.e., a function call interface), or (ii) template (merges component protocols that have periodically executed main functions). For example, the dissemination protocol and proactive membership maintenance protocol can be merged by the template rule so that one protocol's message is piggybacked on top of the other, saving on communication.

An application of the type App-1 uses the DHT through a {get(object),put(object)} API [6]. This is used in a large class of DHT-based p2p applications including CFS, PAST, Squirrel, Ivy, etc. App-2 describes a model where the application is "pushed down into the DHT layer" in the interests of protocol efficiency. This fits the cooperative web caching application built over the Kelips DHT [14]. The App-2 model may recode some blocks from the DHT.

The above two composition rules are borrowed from a different methodology for probabilistic protocols [18]. Like that methodology, Figure 1 is also *automatable*, i.e., the high level language (PPCL) and toolkit described in [18] can be used to generate code for DHTs and applications on the fly. Figure 2 shows salient parts of the PPCL specification for the methodology of Figure 1.

The presented methodology thus encapsulates the philosophy behind a class of DHTs and p2p applications. Besides this retroactive use, it has significant progressive and auxiliary advantages.

In Section 4, we evaluate one progressive use of the methodology. Figure 1 showed an option of two different designs for membership maintenance protocols (heartbeating or random ping-based [7]) - this is an example of auxiliary methodologies brought to bear. Figure 3 shows possible benefits from four other existing methodologies. Other properties such as security may also be introduced through auxiliary composable security protocols.

4   Experimental Results

One benefit of a methodology is the assistance in exploring protocol design space. Consider a designer asked to build a Pastry-based application for an existing (legacy) overlay, where the neighbor relation among nodes may not obey the Pastry neighbor relation, but is specified by a different application. For instance, this application might be a legacy one that closely relies on a pre-existing membership protocol, but desires to augment itself with Pastry-like search capabilities.

Viewed in the context of Figure 1, the designer's challenge reduces to – the designer can only change the implementation of the App-1, Insert, Lookup and Routing blocks, but not the rest of the design.

Figures 4(a-c) show that if the overlay is just chosen by each node selecting neighbors uniformly at random, the default Pastry routing is inappropriate due to its low query success rate. However, our methodology allows us to explore

```
/* Blocks to be called by App-1. */
component {
     file = dht.c;
     function = char *get(void *);
} lookup;
component {
     file = dht.c;
     function = char *put(void *, char *);
} insert;
/* Blocks with call-in functions. */
component {
     file = overlay.c;
     function = get_neighbor(int dest);
} overlay;
...
/* Compositions -- Augmentation. */
augment {
     target = lookup;
     replace(int forward_search(void *data), overlay);
} lookup-overlay;
     // replace each call to forward_search in lookup
     // with call to main function of overlay
...
/* Periodically Executed Blocks. */
component {
     file = swim.c;
     function = void run();
} membership;
component {
     file = diss.c;
     function = void run();
} dissemination;
/* Composition -- Merging membership and dissemination. */
...
template {
     component = membership;
     component = dissemination;
     match (int send_member_update(),
            int send(),
            int send(),
            piggyback);
     match (int process(char *recv_msg),
            int process(char *recv_msg),
            int process_join(char *piggy_backed, char *recv_msg),
            unpiggyback);
     match (recvmsg, in_buff, recvmsg);
} membership-dissem;
// Merges the main functions of membership.c and dissemination.c. For each
// match statement (a(), b(), c(), d()), a call to a() in membership and b()
// in dissemination are merged, top-down in the code, and replaced with a
// call to c(). d() is an optional user-defined function that merges arguments
// to a() and b() for input into c().
```

Fig. 2. PPCL code (snapshot) for automating the Methodology of Figure 1:
This specification is derived from existing source code for individual components. The toolkit in [18] generates code from this and the component source code.
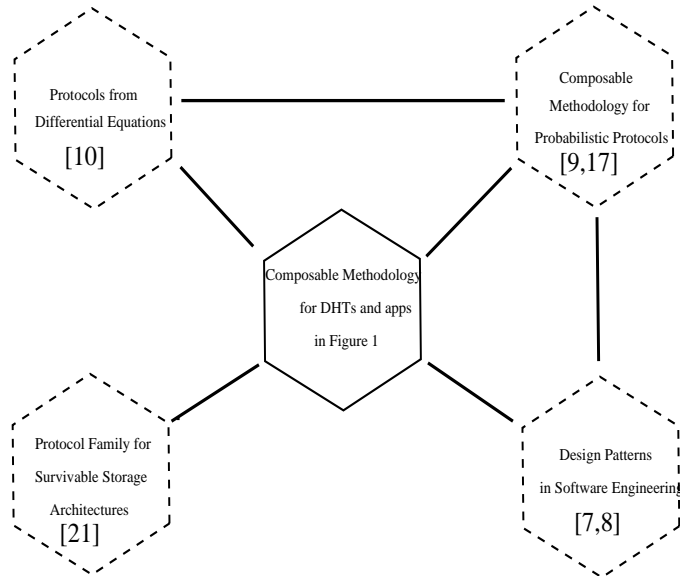
Fig. 3. The Power of Auxiliary Methodologies: *Other methodologies that serve as auxiliary methodologies for (i.e., either benefit, or benefit from) each other as well as for the composable methodology of Figure 1.*

several alternative designs by modifying the *Routing* block of Figure 1. We consider one such new routing protocol called *multiflow routing.* Here, a node first finds all its neighbors with longer prefix matches to the destination ID than its own. The query is then forwarded to only the top 2 among these neighbors, with such multi-forwarding limited to 3 times per query per route.

The plots show that multiflow routing outperforms prefix routing on query success rate (close to 100%) and latency. The traffic increase is moderate (a factor of about 3) at 10% membership list size. This tradeoff would be acceptable to an overlay running in medium-sized groups, and where the application requires reliable querying.

5 Our Current Work

We are currently working on both new innovative and composable methodologies. Below, we briefly summarize two methodologies we have developed, along with some of their uses.
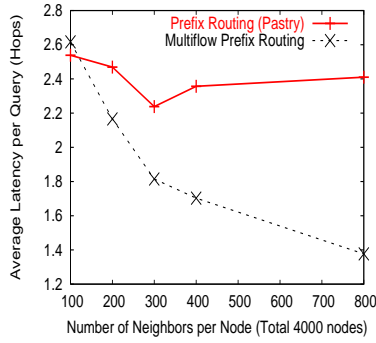
5.1 Innovative Methodology – Generating Self-* Protocols from Differential Equations

We have discovered an innovative methodology that translates sets of differential equations into *equivalent* distributed protocols [11]. An equation set generates
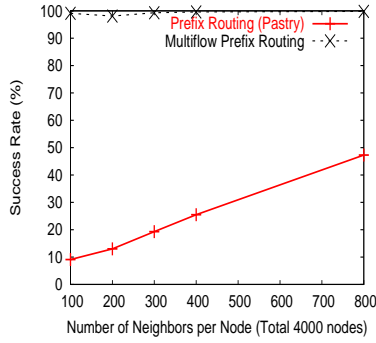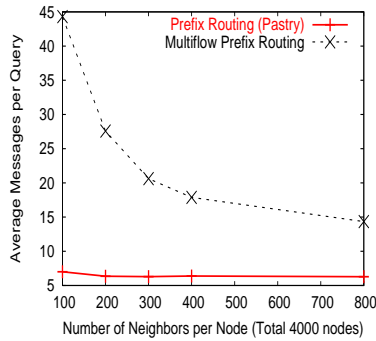
(a) *Query Success Rate.*

(b) *Query Reply Latency.*

(c) *Messages per Query.*

Fig. 4. Default Pastry Routing versus new Multiflow Routing for random overlays: *Prefix routing is default Pastry routing; multiflow prefix is the new proposed routing protocol within Figure 1. Each point on the plot is averaged from 1000 queries over 10 different random overlays.*

a state machine, with each variable mapped to a state, and terms mapped to protocol actions. Stable equilibrium points in the original equations map to self-stabilizing behavior (fractions of nodes in respective states), and simplicity of terms maps to scalable communication.

In brief, consider a system of differential equations in the form $\dot{\bar{X}} = \frac{d\bar{X}}{dt} = \bar{f}(\bar{X})$, where $\bar{X}$ is a vector of variables, f is a vector of $|X|$ functions, and the left hand sides denote each of the variables differentiated with respect to time t. An example with $X = \{x, y, z\}$ is:

$$\dot{x} = -\beta xy + \alpha z$$
$$\dot{y} = \beta xy - \gamma y$$
$$\dot{z} = \gamma y - \alpha z \tag{1}$$

Here, α, β, γ are parameters lying in the interval $[0, 1]$. When the methodology of [11] is applied to the above equation, the protocol generated is *equivalent* to the original equation system, viz., *the fractions of processes in different states in the distributed system, at equilibrium, is the same as the values of the variables when the original equations are in equilibrium.* We provide here a summary of the translation techniques and the uses of the derived protocols - for more details, the reader is encouraged to refer to [11].

Summary of Translation Techniques: In order to translate a system of differential equations $\dot{\bar{X}} = \frac{d\bar{X}}{dt} = \bar{f}(\bar{X})$, the right hand sides of these equations are required to each be a sum of polynomial terms $\pm T = \pm c_T \Pi^{y \in X} y^{i_{y,f_x,T}}$ ($0 \leq c_T \leq 1$, $i_{y,f_x,T}$ non-negative integers), each negative term occurring in $\dot{x}$ should have a power of x that is ≥ 1, and terms should be *pair*-able into matching pairs of positive and negative terms so that each pair sums to zero. The translation methodology then creates a state machine which has one state per original variable in the set $\bar{X}$. It also creates an action for each negative:positive term pair – this action is executed periodically (once every protocol period, protocol period duration fixed) when a process is in the state x, where $\dot{x}$ contains the negative term. Without loss of generality, suppose that the positive part of the pair occurs in $\dot{z}$.

Translating a simple term pair such as $-c.x:+c.x$ requires the process in state x to flip a coin with heads probability c, and transition to state z only if this falls heads. Translating other polynomial terms is achieved by not only flipping a local coin but also sampling a random selection of other processes, and deciding whether to transition or not based on their states. For instance a term pair $-T : +T$ with $-T = -x.y^2$ can be translated into an action where a process in state x periodically (once every protocol period) samples two other processes selected uniformly at random from across the group. If both sampled processes happen to be in state y, then p transitions into state z.

Equation systems that do not satisfy the above conditions (e.g., in term matching, or in the form of the basic equation system) may need to be normalized or rewritten. Several equation rewriting techniques are discussed in [11].

Emergent Properties of Derived Protocols and their Uses: The methodology retains the same equilibrium points as in the original differential equation, thus providing the derived protocol with *self-stabilizing behavior* around the stable equilibrium points. The stochastic behavior of the differential equations translates into *fault-tolerance, churn-resistance*, and other interesting properties such as *attacker resilience*, as described below. Finally, the simplicity of the equations is reflected in the simplicity and *constant (hence scalable) communication overhead* of the derived protocol.

Equations (1) above in fact represent the survival of endemic diseases (such as influenza) in a fixed-size human population, with x, y, z respectively the fractions of receptives, infected, and immune individuals. The protocol derived from it is a new *dynamic and migratory replication model.* In this dynamic and migratory replication scheme, once a given file is inserted into a distributed group of computer hosts connected in an overlay, the emergent behavior of the protocol ensures that the file has a small number of replicas moving continuously among the hosts in the distributed system (only "infected" hosts store a replica). Without using too much network bandwidth, this scheme ensures *attacker-resilience* – an attacker only has very short windows (typically tens of seconds) to guess the exact number and location of the current replicas for a given file. This dynamic and migratory replication model is currently being used to design a new distributed file system called "Folklore".

The above methodology can also be used to design a new protocol for majority voting in large-scale distributed systems (derived from a model of biological competition among ecosystem species) [11].

The protocols generated by this methodology are similar to Complex Adaptive Systems, and can be considered as a type of "Emergent Thinker" [4].

This differential equation translation methodology has been incorporated into a design toolkit called *DiffGen*, which enables a designer to input differential equations (in a Mathematica-like format), and outputs compilable and deployable C code for the equivalent protocol.

5.2 Composable Methodology - Probabilistic Protocols for Large-Scale Distributed Systems

In [18], we have developed a toolkit called PPCL that automates a composable methodology for designing probabilistic protocols for large-scale distributed systems. This toolkit extends and implements the methodology of [10]. Multicast, aggregation, leader election, failure detection, membership, are some of the problems solved.

The generated probabilistic protocols are self-adaptive to failures. The protocols have very high reliability, per-process overheads that vary from being either independent of or polylogarithmically dependent on the number of processes in the distributed group, and protocol completion times that are also polylogarithmic.

The methodology consists of seven classes of building blocks, each of which is either a protocol or a strategy: epidemics, distributed ping, tree dissemination,

recovery and committee selection are protocols while non-protocol strategies include weak overlays, and topology-aware probability distribution functions. Each building block has well-understood scalability and reliability properties. The composition techniques are either simple *augmentations* or *template compositions*. For example, an epidemic protocol that selects targets uniformly at random can be made topologically aware by augmentation with a network round trip-time-based probability distribution function for target choice. Another example is that a failure detector protocol and a multicast protocol can be composed through a template composition (see Figure 2) to generate a protocol for decentralized membership maintenance.

For such composable design methodologies, one has to ensure that performance properties such as correctness, reliability, scalability are either inherited or preserved to an extent, in spite of the composition. For instance, in the above methodology, template compositions inherit both correctness properties (e.g., eventual delivery of multicasts) as well as performance properties (e.g., overhead, multicast latency, reliability) from components. Augmentation inherits correctness, while preserving performance to a large extent – latency, overhead, etc. degrades by a factor that is typically polylogarithmic in system size.

The reader is refered to [10, 18] for more details.

6   Summary

In this paper, we have presented a new taxonomy for design methodologies, and a new concrete methodology for such self-* distributed systems as DHTs and p2p applications. We have argued that design methodologies for self-* distributed systems can capture the designer's mindframe and the philosophy behind a class of distributed protocols, thus enabling both exploration of the protocol design space and systematic protocol reuse. Many methodologies can be automated to generate ready-to-deploy code.

Methodologies are understood by theoreticians, practitioners, researchers, and vendors alike, e.g., terms familiar to all these communities include "composability" [21, 22]. Cultivation of design methodologies is absolutely essential for systematic protocol design to emerge for self-* distributed systems.

References

1. Proc. 1st-3rd IPTPS, 2002-2004.
2. Arvind. Bluespec: A language for hardware design, simulation, synthesis and verification. In *Proc. MEMOCODE*, page 249, Jun. 2003.
3. B. Bershad and S. Savage et al. Extensibility, safety and performance in the SPIN operating system,. In *Proc. ACM SOSP*, pages 267–284, Dec. 1995.
4. S. Camorlinga and K. Barker. The emergent thinker. In *Proc. SELF-STAR: Intnl. Workshop on Self-* Properties in Complex Information Systems*, May-Jun. 2004.
5. Computing Research Association (CRA). Grand Research Challenges in Distributed Systems. http://www.cra.org/reports/gc.systems.pdf.
6. F. Dabek, B. Zhao, and P. Druschel et al. Towards a common API for structured peer-to-peer overlays. In *Proc. IPTPS*, pages 33–64, 2003.
7. A. Das, I. Gupta, and A. Motivala. SWIM: Scalable Weakly-consistent Infection-style process group Membership protocol. In *Proc. DSN*, pages 303–312, 2002.
8. E. Gamma, R. Helm, R. Johnson, and Vlissides J. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1st edition, 1995.
9. B. Garbinato and R. Guerraoui. Using the strategy design pattern to compose reliable distributed protocols. In *Proc. USENIX Conf. Obj.-Or. Tech. and Sys.*, pages 221–232, Jun. 1997.
10. I. Gupta. *Building Scalable Solutions to Distributed Computing Problems using Probabilistic Components*. PhD Thesis, Dept. of Computer Science, Cornell University, Jan. 2004.
11. I. Gupta. On the design of distributed protocols from differential equations. In *Proc. ACM PODC*, pages 216–225, 2004.
12. A. Iamnitchi, M. Ripeanu, and I. Foster. Locating data in (small-world?) p2p scientific collaborations. In *Proc. IPTPS*, pages 232–241, 2002.
13. E. Kohler and R. Morris et al. The Click modular router. *ACM Tr. Comp. Sys.*, 18(3), Aug. 2000.
14. P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. In *Proc. ACM Wshop. SSRS*, Oct. 2003.
15. G. S. Manku. Balanced binary trees for id management and load balance in distributed hash tables. In *Proc. ACM PODC*, pages 197–205, 2004.
16. A. Spector. Plenary Talk. ACM SOSP, 2003.
17. B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. In *Wshop. Planning for Web Serv.*, pages $28 - 35$, 2003.
18. N. Thompson and I. Gupta. A composable methodology for proactive distributed protocols, Sept. 2004. TR UIUCDCS-R-2004-2490,.
19. R. van Renesse, S. Maffeis, and K. P. Birman. Horus: a flexible group communications system. *CACM*, 39(4):76–83, April 1996.
20. J. Ventura, J. Rodrigues, and L. Rodrigues. Response time analysis of composable micro-protocols. In *Proc. 4th IEEE OORTDC*, pages 335–342, 2001.
21. S.-H. Wu and S. A. Smolka et al. Composition and behaviors of probabilistic I/O automata. *TCS*, 176(1-2):1–38, Apr. 1997.
22. J. Wylie and G.R. Goodson et al. A protocol family approach to survivable storage infrastructures. In *Proc. FUDICO*, 2004.
23. G. Xie and J. Zhan et al. Routing design in operational networks: a look from the inside. In *Proc. ACM SIGCOMM Conf.*, pages 27–40, 2004.