Compositional Modeling for Data-Centric Business Applications

Ethan K. Jackson and Wolfram Schulte

Microsoft Research,
One Microsoft Way, Redmond, WA
{ejackson,schulte}@microsoft.com

Abstract. Data-centric business applications comprise an important class of distributed systems that includes on-line stores, document management systems, and patient portals. However, their complexity makes it difficult to design and implement them. We address these issues from a model-driven perspective by developing a formal, compositional, and domain-specific set of abstractions for the specification and analysis of data-centric business applications. Our technique allows us to formally analyze the specified system at design time; in particular we can analyze whether the system is resilient to abnormal conditions, i.e. that key system invariants can always be re-established.

1 Introduction

Data-centric business applications comprise an important class of distributed systems that includes on-line stores, document management systems, and patient portals. However, many foundational issues make the design of correct business applications a non-trivial problem: Implementation technologies are diverse and problematic, e.g. the security of Web 2.0 applications is flawed [1]. Difficult non-functional requirements, such as privacy [2], are often overlooked. Anecdotally, even mature and well-tested distributed systems still fail under unexpected conditions [3].

A number of approaches have been suggested to address these issues: *Model-driven architecture* (MDA) [4] attempts to disentangle the implementation platform from the business logic, *enterprise patterns* [5] distill isolated kernels of programmer wisdom, and formal *work-flow models* [6] focus on arrangement and communication of processing tasks. Each approach addresses some aspect of the overall design problem, but no existing approach combines the specification of the business logic, its distributed implementation, and its formal analysis.

We address these issues from a model-driven perspective by developing a formal, compositional, and domain-specific set of abstractions for the specification and analysis of data-centric business applications. Our contributions are:

– We provide a new specification technique for modeling distributed and data-centric business applications. This is accomplished by the specification and

composition of three models: A *data model* (Section 2) enumerates the essential data and data invariants of the system. An *operation model* (Section 3) characterizes the set of data manipulation operations. A *connectivity model* (Section 4) defines the agents of the system, the information flow between agents, and the bindings of data to agents, thereby generalizing data and operations over networks.
- We give these models a formal semantics based on term algebras and inference over terms using Horn logic extended with stratified negation [7]. With this framework we can characterize the unstable states, which are those distributed states that temporarily violate invariants of the data model.
- We show in Section 5 that for any system specified with BAM there exists a finite set of equivalence classes of unstable states of finite size. This allows the application of finite model checking to determine if a system is self-stabilizing; i.e. that a BAM system has adequate operations for reestablishing invariants.

We discuss related work in Section 6 and future work in Section 7.

1.1 Running Example: A Document Management System

The high-level requirements of data-centric applications are often straightforward. For example, the requirements for a document management system are:

1. Authorized users can view, create, and modify their documents, from a local *client*.
2. Authorized users can manage their documents, even if not connected to a *server*.
3. One or more *servers* synchronize with *clients* to record the latest versions of documents.

This example, while simple, still illustrates some important characteristics of data-centric business applications. First, the key data elements are readily apparent – e.g. servers contain *copies*. Second, data can be manipulated by some small set of (simple) operations – e.g. clients create documents. Third, heterogeneous agents act upon the data, and different agents have different capabilities – e.g. servers must synchronize with clients, while clients are free to modify the documents they own.

2 Data Model

BAM data models capture the data states of a business application using *metamodels* employing a notation similar to *UML class diagrams* [8]. Figure 1 shows a data model for our document management system. For example, the Document construct contains two pieces of information: The user who created it an its title. A Snapshot contains a set of DocumentCopy items. The consistency between Documents and DocumentCopies are specified with a set of formal invariants; these appear in the gray boxes on the right-hand side. (The invariants will be described in more detail later.)

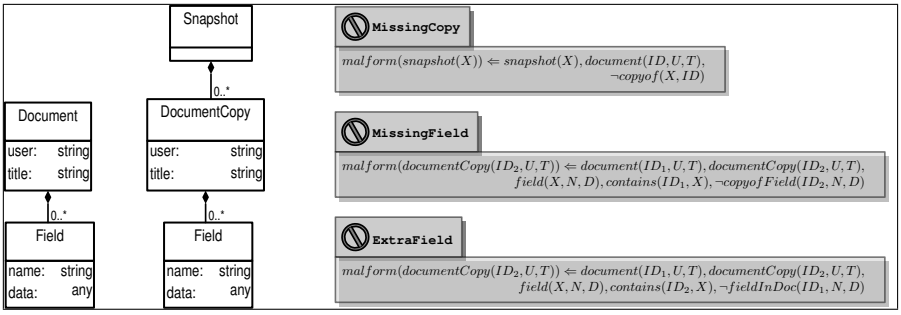

Fig. 1. Data model for document management system

Our terminology and formalization of metamodels and model transformations builds on previous work [9]. We briefly repeat necessary concepts below; for more details consult the reference.

Formally, a data model expresses a four-tuple $D = \langle \Upsilon, \Upsilon_C, \Sigma, C \rangle$ called a *domain*, where Υ is a finite signature representing the data constructs, Υ_C is a finite signature for representing derived properties, Σ is a set of values, and C is a set extended Horn axioms defined over $\Upsilon, \Upsilon_C, \Sigma$ for deriving properties. These axioms are used to capture data invariants. The set of all possible data states is the *powerset* of the *term algebra* over Υ generated by Σ. This is written $\mathcal{P}(\mathcal{T}_\Upsilon(\Sigma))$. A member $s \in \mathcal{P}(\mathcal{T}_\Upsilon(\Sigma))$ is a concrete instantiation of data, and describes the state of the management system at some time.

Returning to the example, Figure 2 shows a visualization for a particular state s. It contains two documents, entitled EmailToBob and ExpenseReport. The box labeled Snapshot contains several document copies, entitled EmailToBob and Payroll. Under the title of each document/copy is the user who created the document and the associated fields.

Table 1 shows the terms used to encode s, grouped by function symbol. For example, a document is represented by a term $document(ID, User, Title)$, where $document(\cdot,\cdot,\cdot)$ is ternary function symbol and $\{ID, User, Title\} \subset \Sigma$ are values for the unique ID of the document, the user who created the document, and the document's title. The Field, DocumentCopy, and Snapshot occurrences are encoded in a similar fashion. Containment of one occurrence within another is represented by the function symbol $contains(\cdot,\cdot)$, where a term $contains(ID_1, ID_2)$ denotes that the data with ID_1 contains the data with ID_2.

The data invariants are expressed using the help of a standard function symbol, $malform(\cdot)$. A data state s is *inconsistent*, violates invariants, if it is possible to derive any $malform(\cdot)$ terms.

For example, the axiom below explains that it is possible to derive a term of the form $malform(contains(ID_1, ID_2))$ if the state s has terms encoding the containment of a Document within a Snapshot.

$$malform(contains(ID_1, ID_2)) \Leftarrow contains(ID_1, ID_2), snapshot(ID_1), document(ID_2, U, N) \quad (1)$$

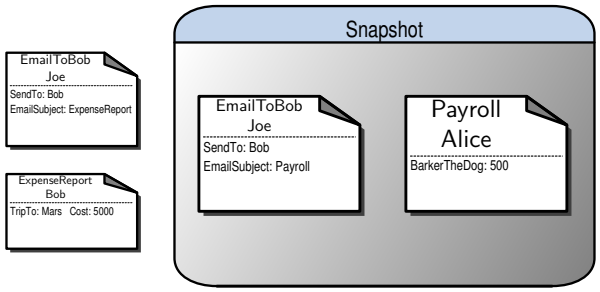

Fig. 2. An example state of a management system

Signature	Terms
$document(\cdot,\cdot,\cdot)$	$document(1, \texttt{Joe}, \texttt{EmailToBob})$, $document(2, \texttt{Bob}, \texttt{ExpenseRep})$
$field(\cdot,\cdot,\cdot)$	$field(3, \texttt{SendTo}, \texttt{Bob})$, $field(4, \texttt{EmailSubject}, \texttt{ExpenseReport})$, $field(5, \texttt{TripTo}, \texttt{Mars})$, $field(6, \texttt{Cost}, 5000)$
$contains(\cdot,\cdot)$	$contains(1,3)$, $contains(1,4)$, $contains(2,5)$, $contains(2,6)$
$snapshot(\cdot)$	$snapshot(7)$
$documentCopy(\cdot,\cdot,\cdot)$	$documentCopy(8, \texttt{Joe}, \texttt{EmailToBob})$, $documentCopy(9, \texttt{Alice}, \texttt{Payroll})$
$field(\cdot,\cdot,\cdot)$	$field(10, \texttt{SendTo}, \texttt{Bob})$, $field(11, \texttt{EmailSubject}, \texttt{Payroll})$, $field(12, \texttt{BarkerTheGuardDog}, 100)$
$contains(\cdot,\cdot)$	$contains(7,8)$, $contains(7,9)$, $contains(8,10)$, $contains(8,11)$, $contains(9,12)$

Table 1. Abstraction of document management state as a set of terms

Returning to the original data model, indeed we see that Documents cannot be contained in Snapshots. This is a simple invariant, but the data model can capture more complex invariants that are implied by the high-level requirements. These invariants a shown on the right-hand side of Figure 1. For example, we require that a Snapshot should contain a copy of any known Document. The invariant labeled MissingCopy expresses this

$$malform(snapshot(ID_1)) \Leftarrow snapshot(ID_1), document(ID_2, U, T), \neg copyof(ID_1, ID_2) \tag{2}$$

using the auxiliary axiom:

$$copyof(ID_1, ID_2) \Leftarrow snapshot(ID_1), document(ID_2, U, T), documentCopy(ID_3, U, T), contains(ID_1, ID_3) \tag{3}$$

Note that this expresses the synchronization between documents and copies in a data-centric manner, without explaining the protocols and services necessary to implement this synchronization. The remaining invariants, MissingField and ExtraField, define those states with discrepancies between the fields of documents

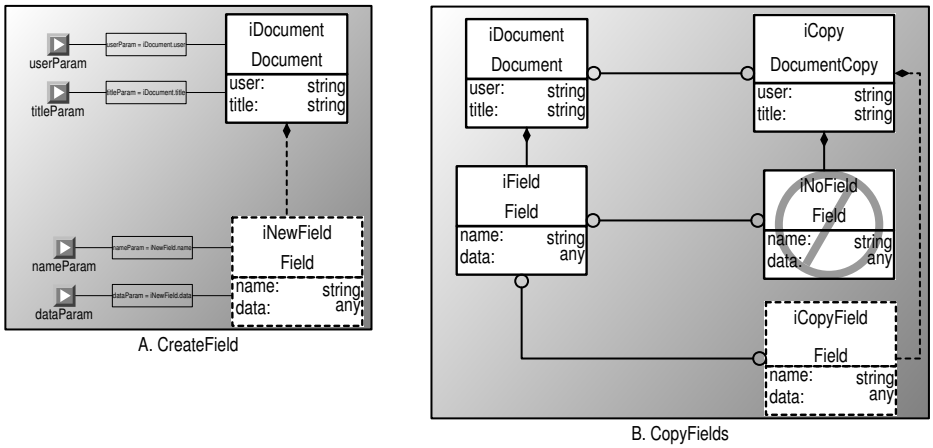

Fig. 3. Two BAM operations used by the document management system.

and copies. For notational convenience, let $models(D)$ denote the set of data states that satisfy invariants.

3 Operation Model

Data-centric business applications inevitably require some basic operations on data, and the BAM *operation model* captures these essential operations as *model transformations*. A model transformation changes the current data state s by adding new terms to s and/or deleting existing terms from s.

Formally, a model transformation $\lambda = (t_\lambda^+, t_\lambda^-)$ is comprised of two sets of Horn axioms. The axioms of t_λ^+ derive the terms that should be added to s, and the axioms of t_λ^- derive the terms that should be removed. If the data state is s, then an operation λ changes the state to s_λ according to the state update equation:

$$s_\lambda = \left(s \cup [\![s]\!]^{t_\lambda^+} \right) - [\![s]\!]^{t_\lambda^-} \tag{4}$$

where $[\![\ ]\!]^t$ is the map that takes a set of terms s to the set of new terms derivable from s by axioms t.

3.1 Parameterless Operations

Figure 3.B shows a *parameterless* BAM operation, called CopyFields, that finds all the fields contained in a document that are not contained in its corresponding copy. This operation then adds these missing fields to the copy so that it is consistent with the original document. Formally, the CopyFields operation has

the following axioms in t^+_λ:

$$field(new(ID_3), N, D), contains(ID_2, new(ID_3)) \Leftarrow document(ID_1, U, T), \\ documentCopy(ID_2, U, T), field(ID_3, N, D), \\ contains(ID_1, ID_3), \neg copyofField(ID_2, ID_3) \quad (5)$$

$$copyofField(ID_2, ID_3) \Leftarrow documentCopy(ID_2, U, T), field(ID_3, N, D), \\ contains(ID_2, ID_3) \quad (6)$$

where $new(\cdot)$ creates new identifiers that are not currently in the state s. [1]

3.2 Parameterized Operations

Some operations must take input from the external environment. A user U may wish to create a new document with title T that does not exist in the current state. This issue can be addressed by parameterizing the transformation axioms of λ to create a family of concrete operations. A *parameterized transformation* $\lambda(p_1, p_2, \ldots, p_n)$ has n parameters and is concretized by assigning each $p_i = \sigma_i \in \Sigma$. The concrete transformation is a model transformation formed by replacing every occurrence of p_i with σ_i in the transformation axioms of $\lambda(p_1, \ldots, p_n)$.

The CreateField operation in Figure 3.A illustrates a parameterized operation. This operation has four parameters: p_u, p_t, p_n, p_d. The parameters p_u, p_t are used to find an existing document created by user p_u titled p_t. If such a document is found, then a new field named p_n with data p_d is created within this document. The parameterized axiom for CreateField is as follows:

$$field(new(ID_1), p_n, p_d), contains(ID_1, new(ID_1)) \Leftarrow document(ID_1, p_u, p_t), \quad (7)$$

For example, the operation CreateField(Bob,ExpenseReport,Taxi,50) adds a field to Bob's expense report indicating that he spent 50 dollars on a taxi.

4 Connectivity Model

The data and operation models specify the core functionality of data-centric business applications. The third BAM model, called the *connectivity model*, describes how data an operations interact across logical networks. The formal semantics of BAM uses this information to generalize the satisfaction of data invariants and calculation of state updates over arbitrary network topologies.

Formally, a connectivity model is a triple $G_{conn} = \langle A, F, \rho \rangle$ where A is the set of agent types, $F \subseteq A \times A$ is the information flow between agent types, and $\rho : A \to \mathcal{P}(\Upsilon)$ is a mapping from agent types to function symbols of the data model. The mapping ρ provides an access control mechanism, i.e. agents of type $a \in A$ can only access data of types $\rho(a)$. This access control respects

[1] Formally, $new(\cdot)$ is a state-dependent bijection $new : \Sigma \to \Sigma$ such that for each element σ appearing in s, $new(\sigma)$ is not in s.

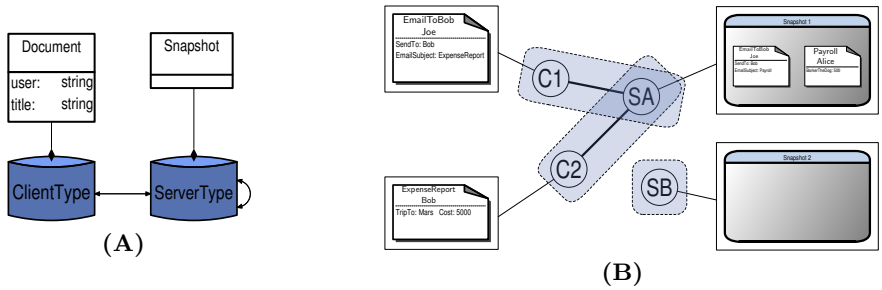

Fig. 4. (A) Connectivity model of document management system (B) Example of a composite state

containment relationships between data: If $f \in \rho(a)$ and occurrences of f can contain occurrences of g then $g \in \rho(a)$.

Figure 4.A shows the connectivity model for the document management system. There are two types of agents: ClientType and ServerType agents. Each agent has some data structures bound to it. For example, only ClientType agents contain Document items, while Snapshot items are only present in ServerType agents. The double-headed edges (◀⟶▶) define which types of agents can communicate with each other, e.g. ClientType and ServerType agents can communicate. Notice that information flow must be explicitly defined, even for flow between agents of the same type. In this example information can pass between ServerType agents, but not ClientType agents. The connectivity model for the document management system is:

$$
\begin{aligned}
&A = \{ClientType, ServerType\},\\
&F = \{(ClientType, ServerType), (ServerType, ServerType), \ldots\}\\
&\rho(ClientType) = \{document(\cdot,\cdot,\cdot), field(\cdot,\cdot,\cdot), contains(\cdot,\cdot)\}\\
&\rho(ServerType) = \{snapshot(\cdot), documentCopy(\cdot,\cdot,\cdot), field(\cdot,\cdot,\cdot), contains(\cdot,\cdot)\}
\end{aligned}
$$

4.1 Composite States

Instead of viewing the system behaviors as a sequence of transitions through data states ($s_0 \rightarrow s_1 \rightarrow \ldots$), we can now view the system as transitioning through *composite states* ($\mathcal{C}_0 \rightarrow \mathcal{C}_1 \rightarrow \ldots$), which take into account the distributed state of the network.

Let D be a domain (as given by the data model) and G_{conn} be the connectivity model, then $\mathcal{C}(D, G_{conn})$ is a composite state parameterized by D, G_{conn}.

Definition 1. *A composite state $\mathcal{C}(D, G_{conn}) = \langle V, E, type, \delta, \Lambda \rangle$ is a quintuple where:*

1. *(V, E) is a finite undirected graph, where V is the set of agents and E is the information flow between agents. We call this the connectivity state.*
2. *$type : V \rightarrow A$ is a mapping from agents to agent types, denoting the type of each agent.*

3. $\delta : V \to \mathcal{P}(\mathcal{T}_\Upsilon(\Sigma))$ *is mapping from agents to data states. If* $u \in V$ *is an agent, then* $\delta(u)$ *is its local data state. We call* δ *the global data state.*
4. Λ *is the set of operations available from the current state; it is called the operation state.*

In order for a structure $\mathcal{C}$ to be a valid composite state it must respect the various models that parameterize it. First, the connectivity state must respect the connectivity model G_{conn}. This holds if the logical connectivity between agents respects the information flow:

$$\forall (v, u) \in E,\ (type(v), type(u)) \in F \tag{8}$$

Also, the local data state of each agent must respect the access control of the connectivity model:

$$\forall v \in V,\ \delta(v) \subset \mathcal{T}_{\rho(type(v))}(\Sigma) \tag{9}$$

In other words, the terms of the data state are a subset of a smaller term algebra created over the smaller signature $\rho(type(v))$. Finally, the operations of the *operation state* must defined over the signature Υ and alphabet Σ of the domain D.

Composite states have a preorder $\leq$ with respect to the data contained by the vertices. We say that $\mathcal{C}' \leq \mathcal{C}$ if the states have the same topology, but each vertex in $\mathcal{C}'$ contains the same or less data than the corresponding vertex in $\mathcal{C}$.

Definition 2. *Let* $\leq$ *be an ordering over composite states where* $\mathcal{C}' \leq \mathcal{C}$ *if:*

1. $(V', E') \cong (V, E)$*; the graphs are isomorphic as witnessed by* π*.*
2. $type(\pi(v)) = type(v)$*; the types are preserved between states.*
3. $\delta(\pi(v)) \subseteq \delta(v)$*; vertices in* $\mathcal{C}'$ *contain the same or less data.*

This ordering over composite states will become important in the latter sections.

4.2 Stability in Composite States

The invariants of the data model must also be extended to composite states. One solution might be to consider invariants over the union of the global data state $\bigcup_{v \in V} \delta(v)$. Figure 4.B shows a composite state for the document management system that illustrates why this approach fails. This state has two ServerAgent nodes SA, SB and two ClientAgent nodes C1, C2. Information flows between SA and C1, as well as between SA and C2. Attached to each vertex v is its data state $\delta(v)$. Consider server SB, which does not contain any copies of documents. If invariants were checked over the union of the data state, then SB would fail the requirement that snapshots contain copies of existing documents. However, no information can flow between the clients C1, C2 and SB; this should alleviate SB from the burden of satisfying this invariant. Information flow naturally induces a relaxation of the invariants. This suggests that invariants should be checked over the data models formed by connected components. However, this assumes a *transitive* flow of information that usually does not hold. For example, the two

clients are not connected and will not be able to observe each others documents, even though they are transitively connected through the server.

We propose evaluating invariants over the *maximal cliques* of the connectivity state. These maximal cliques are the maximal subgraphs where information can flow between all agents in the subgraph. [2] A composite state $\mathcal{C}$ is *stable* if the data state formed by each maximal clique is consistent (i.e. satisfies the invariants):

$$\forall m \in \mathbf{maxcliques}(\mathcal{C}),\ (\bigcup_{v \in m} \delta(V)) \in models(D) \tag{10}$$

Figure 4.B shows the maximal cliques outlined in blue. Under this interpretation SB does not cause the state to be unstable. Mathematically, our maximal clique semantics partitions the network into symmetric subgraphs where information flow is universal within the subgraph. We shall see that this symmetry provides a powerful foundation for reasoning about distributed systems. Returning to Figure 4.B, notice that this state is still unstable because SA does not have a copy of the ExpenseReport document in client C2. (See Figure 2 for a larger view of the snapshot on server SA.) This situation can be rectified by applying an operation that copies new documents from the client to the server. However, this requires generalizing operations over arbitrary topologies, which we describe in the next section.

4.3 Operations over Composite States

Two issues must be addressed when operations are performed over arbitrary topologies. First, how many nodes in the network are involved in an operation? Second, how is distributed data state aggregated to form the input to an operation, and how are the effects of an operation propagated across nodes? To address these issues, we define the notion of an *information extent*.

The information extent of an operation λ, written **iext** λ, describes the data types it must access to operate. In particular, **iext** λ is a multiset of function symbols found in the transformation axioms. [3] The cardinality of a symbol f in **iext** λ determines the lower bound on the number of distinct network nodes that participate in the distributed operation.

For example, the operations of Figure 3 have information extent:

$$\begin{aligned} &\textbf{iext } \mathsf{CreateField} = \{document(\cdot,\cdot,\cdot), field(\cdot,\cdot,\cdot)\} \\ &\textbf{iext } \mathsf{CopyFields} = \{document(\cdot,\cdot,\cdot), documentCopy(\cdot,\cdot,\cdot), field(\cdot,\cdot,\cdot)\} \end{aligned} \tag{11}$$

where each function symbol has cardinality 1.

[2] Formally, a clique is a set $m \subseteq V$ such that $\forall v, u \in m,\ (v,u) \in E$. The clique m is maximal if for every $w \in (V - m)$ then $m \cup \{w\}$ is not a clique.

[3] By a multiset, we mean a set X equipped with a function $\# : X \to \mathbb{N}$ assigning a positive non-zero cardinality to each element in X. We write $\#(f, X)$ to denote the cardinality of function symbol f in multiset X. Given two multisets X, Y, then $X \leq Y$ if $\forall f \in X,\ (f \in Y) \wedge (\#(f,X) \leq \#(f,Y))$.

A set of nodes V' in the network also has an information extent, which is the multiset of data types collectively accessible by those nodes. (The notation $\uplus$ denotes multiset union.)

$$\textbf{iext } V' = \biguplus_{v \in V'} \rho(type(v)) \tag{12}$$

A node n can execute operation λ, if there is some "reasonable" set of nodes V' such that $(\textbf{iext } \lambda) \leq (\textbf{iext } V')$. We calculate this set by growing a horizon from the node n out through the network until the information requirements are satisfied. This *information horizon* depends on the node n executing the operation, the operation λ, and the composite state $\mathcal{C}$:

$$\begin{aligned}
ihr_0(n, \lambda, \mathcal{C}) &= \begin{cases} \{n\} & \text{if } (\textbf{iext } n) \cap (\textbf{iext } \lambda) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \\
ihr_{i+1}(n, \lambda, \mathcal{C}) &= \left\{ u \in N(ihr_i) \;\middle|\; \exists f \in (\textbf{iext } u) \; \begin{pmatrix} \#(f, \textbf{iext } ihr_i) < \\ \#(f, \textbf{iext } \lambda) \end{pmatrix} \right\} \\
ihr(n, \lambda, \mathcal{C}) &= ihr_\infty(n, \lambda, \mathcal{C})
\end{aligned}$$

where $N(ihr_i)$ denotes the neighbors of the set ihr_i in the connectivity graph. Effectively, the information horizon ihr is the least k such that $ihr_k = ihr_{k+1}$.

The information horizon has several important properties. First, every node in the information horizon must have access to some of the data types accessed by the operation. This prevents a node from calling operations that are not related to data on that node, effectively extending access control to operations. Second, the information horizon stops growing once all the nodes in the horizon satisfy the requirements of the operation. This limits the effects of the operation to a small horizon beyond n.

Figure 5 illustrates some information horizons generated by certain nodes in a composite state. The horizons in the left of the figure result from clients creating new documents. As one might expect, these operations are localized around the clients and do not involve any other nodes in the network. On the other hand, if a client wishes to reconcile documents with a server by calling the CopyFields operation, then this only involves the client and the server (center of the figure). However, if a server calls CopyFields it effects all the clients in its immediate horizon (right-hand side of the figure). All of these behaviors fall naturally from our characterization of the information horizon.

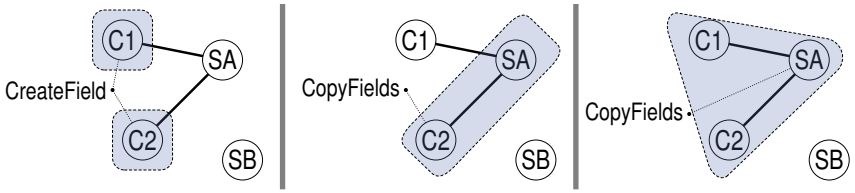

Fig. 5. Information horizons generated by various nodes and operations.

Once an information horizon has been determined, the nodes in the horizon must communicate their data state to aggregate data and calculate the result of an operation. Again, it may be both dangerous (from the security perspective) and inefficient (from the implementation perspective) to aggregate all of the data from all the nodes in the entire horizon. Instead, we reuse the approach from the previous section, and apply multiple instances of the operation over the maximal cliques of the information horizon. This limits communication to those nodes that are logically nearby and have explicit information paths. Let $\mathcal{C}[n, \lambda]$ denote the induced subgraph of $ihr(n, \lambda, \mathcal{C})$ and $\mathbf{maxcliques}(G, n) = \{m | m \in \mathbf{maxcliques}(G), n \in m\}$ be the set of all maximal cliques in a graph G containing a vertex n. For a maximal clique m, the aggregate data state formed by the nodes in the clique is $s(m) = \bigcup_{v \in m} \delta(m)$. Thus, an operation yields a set of state updates similar to Equation 4. For each vertex $v \in ihr(n, \lambda, \mathcal{C})$ the local data state changes to $\delta'(v)$ according to:

$$\delta'(v) = \bigcup_{m \in \mathbf{maxcliques}(\mathcal{C}[n,\lambda],v)} \left(\left([\![s(m)]\!]^{t^+_\lambda} \cup \delta(v) \right) - [\![s(m)]\!]^{t^-_\lambda} \right) \cap \mathcal{T}_{\mathbf{iext}\ v}(\Sigma) \quad (13)$$

This semantics dispatches a copy of λ to each maximal clique. The expression $\left([\![s(m)]\!]^{t^+_\lambda} \cup \delta(v) \right) - [\![s(m)]\!]^{t^-_\lambda}$ denotes λ applied to a maximal clique m containing a vertex v. Finally, the results of the operations are projected against the data types that v is allowed to access: $(\ldots) \cap \mathcal{T}_{\mathbf{iext}\ v}(\Sigma)$.

5 Finitization

Distributed systems exhibit global behaviors that emerge from interacting local agents; predicting these global behaviors is key to validating correctness. In this section we show that the unstable states of the system can be understood in terms of a finite set of composite states. Understanding how instability occurs gives a basis for calculating if the operations are sufficient to correct these instabilities.

We introduce the notion of *acceptors* to represent the ways that invariants can be violated. Formally, an *acceptor* $\alpha = (p, n)$ is a pair such that p and n are sets of terms from some term algebra $\mathcal{T}_\Upsilon(\Sigma)$. The data state s is accepted by α (written $s \models \alpha$) if:

1. There exists some term automorphism π where $\pi(p) \subseteq s$.
2. If $n \neq \emptyset$ then for all term automorphisms π' it holds:
   $\pi(p) = \pi'(p) \Rightarrow \pi'(n) \nsubseteq s$

Returning to our example, an acceptor for MissingField invariant is shown below:

$$\begin{aligned} p &= \{document(7, 2, 3), documentCopy(1, 2, 3), field(8, 9, 10), contains(7, 8)\} \\ n &= \{field(38, 9, 10), contains(1, 38)\} \end{aligned} \quad (14)$$

The numerical arguments to the function symbols are just placeholder constants. Consider once again the data state s in Figure 2 where there exists a document

called EmailToBob containing a field EmailSubject: ExpenseReport. There is a copy of this document, but the copy does not have a field matching this one. Thus, this state is accepted by the MissingField acceptor as witnessed by the following renaming function π:

$$\begin{array}{llll} \pi(1) \mapsto 8, & \pi(2) \mapsto \texttt{Joe}, & \pi(3) \mapsto \texttt{EmailToBob}, & \pi(7) \mapsto 1, \\ \pi(8) \mapsto 4, & \pi(9) \mapsto \texttt{EmailSubject}, & \pi(10) \mapsto \texttt{ExpenseReport} & \end{array} \tag{15}$$

The reader may confirm that $\pi(p) \subset s$ and that there is no renaming function π' that agrees with $\pi(p)$ and has $\pi'(n) \subseteq s$.

The set of acceptors I expresses the invariants as a set of *scenarios*. Each set of positive terms p from any acceptor $\alpha \in I$ describes one scenario of instability. In another words, if we take the data state s to be exactly equal to some p, then this state is unstable. By calculating the set of acceptors I from the invariants, we derive a special set of data states that characterize all the possible forms of instability. If some arbitrary state s' is unstable, then there must be an embedding of some p in s', and the scenarios causing s' to be unstable can be identified. Thus, the acceptors provide a finite description of the unstable data states; we now carry this result over to the more complex composite states. (Note that a term automorphism π (renaming) can be extended to composite states by renaming the data state $\delta(v)$ assigned to each vertex v.)

Composite states complicate analysis because the data state is distributed over the topology, and invariants are relaxed so that they hold over maximal cliques. Our first task is to show that every unstable composite state can be understood in terms of a finite set of scenarios, regardless of the size of the network. We begin by defining a relationship between composite states called a *folding*; this is similar to the familiar *graph homomorphism*:

Definition 3. *Given composite states $\mathcal{C}, \mathcal{C}'$, a folding morphism $\varphi : V \to V'$ assigns vertices of $\mathcal{C}$ to vertices of $\mathcal{C}'$ such that:*

1. *φ is onto.*
2. *$\big[(u = w) \vee (u, w) \in E\big] \Leftrightarrow \big[\big(\varphi(u) = \varphi(w)\big) \vee \big(\varphi(u), \varphi(w)\big) \in E'\big]$*
3. *φ is type-preserving: $type(u) = type'(\varphi(u))$*
4. *φ is data-preserving: $\delta'(u') = \bigcup_{\{u \mid u' = \varphi(u)\}} \delta(u)$.*

We say that $\mathcal{C}'$ is a folding of $\mathcal{C}$ if there exists a folding morphism from $\mathcal{C}$ to $\mathcal{C}'$. Foldings can be combined with the preorder $\leq$ to form a more general preorder $\preceq$ over composite states with varying topologies.

Definition 4. *Given composite states $\mathcal{C}', \mathcal{C}$ then $\mathcal{C}' \preceq \mathcal{C}$ if there exists a $\mathcal{C}''$ where $\mathcal{C}''$ is a folding of $\mathcal{C}$ and $\mathcal{C}' \leq \mathcal{C}''$.*

This ordering is essential to characterizing the types of instabilities that can occur.

In the general case invariants are evaluated against cliques, so we now turn our attention to complete graphs:

Lemma 1. *Let K_n be a composite state where the topology is a complete graph of size n. Furthermore, let K_m be any folding of K_n. In this case, $(s(K_n) \models I) \Leftrightarrow (s(K_m) \models I)$.*

This lemma explains that when the network topology of $\mathcal{C}$ is a complete graph, then the invariants that hold on $\mathcal{C}$ are preserved by folding. It it necessarily the case that any folding of a K_n yields a smaller complete graph of size $m \leq n$. This allows Lemma 1 to be repeatedly applied until no smaller m exists. From the definition of folding morphism, the smallest complete graph that is a folding of K_n must have at least as many vertices as their are types in the composite state K_n. The next lemma shows that this lower bound always exists.

Lemma 2. *Let K_n be a composite state with connectivity that is a complete graph. There exists a folding K_m where m is the number of types in K_n: $m = |\{type(u)|u \in V\}|$. There is no composite state $\mathcal{C}'$ with fewer vertices or fewer edges for which $\mathcal{C}'$ is a folding of K_n.*

Combining Lemmas 1 and 2 we conclude that if any complete graph fails an invariant (or is accepted by some $\alpha \in I$), then it can be folded into a minimum complete graph that also fails the invariant. Since these minimum complete graphs are determined only by the number of types in the system, they can be finitely enumerated.

Let $\emptyset \subset T \subseteq A$ be a set of agent types, then $K(T)$ denotes the set of all completely connected composite states with exactly one vertex for each type in T. Each $K(T)$ still contains an infinite number of states due to the data states. However, not all of these data states are important from the perspective of invariants. The only interesting composite states are those with aggregated data states $s(K_m) = p$ for some $\alpha \in I$. Let $K(T, s)$ be the set of all composite states K_m in $K(T)$ with aggregated data state $s(K_m) = s$:

Lemma 3. *Given a finite set of acceptors I and a finite set of agent types A, then the set I_K is finite.*

$$I_K = \bigcup_{T \subseteq A} \left\{ \mathcal{C} \in K(T, p) \;\middle|\; \exists (p, n) \in I \right\} \tag{16}$$

This set contains all completely connected composite states that are topologically minimal and have data states from some scenario in I. The number of acceptors $\alpha \in I$ is finite, each p is a finite set of data, and every minimal topology contains

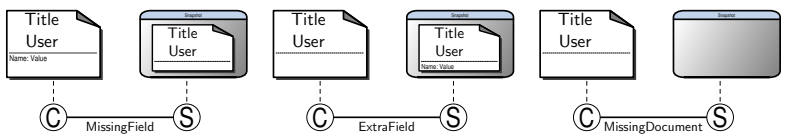

Fig. 6. Key unstable scenarios generated from BAM model of document management system.

a finite number of nodes. Putting these together, there are only a finite number of ways that each p can be split across a fixed topology so I_K must contain a finite number of composite states.

In the general case of an arbitrary unstable composite state $\mathcal{C}$ there must be some maximal clique m where the data state $s(m)$ over the clique violates an invariant (Equation 10). This maximal clique is a completely connected composite state K_n and so it has a folding onto some $K_l \in K(T)$ where T is the set of types that appear in the clique. Furthermore, the data state $s(m)$ must embed some p from an acceptor; K_l also embeds this p as folding preserves data. Therefore, there is some scenario $i \in I_K$ and some term automorphism π (i.e. renaming of constants) such that $\pi(i) \preceq m$.

Theorem 1. *If $\mathcal{C}$ is an unstable composite state, then there exists a maximal clique m in $\mathcal{C}$ such that $\pi(i) \preceq m$ for some $i \in I_K$ and some renaming π.*

This result shows that I_K contains the fundamental ways that an arbitrary state can be unstable, and these can be calculated automatically from a BAM model.

Figure 6 shows the key scenarios generated from the BAM model of the document management system. The MissingField scenario occurs when a document has a field not found in its copy. Similarly, the ExtraField scenario occurs when a copy has a field not found in the original document. Finally, the MissingDocument scenario occurs when a snapshot does not have a copy of a document.

6 Related Works

This work uses the techniques of *model-based design* [10], which constructs complex systems through formal domain-specific abstractions and code synthesis techniques. Model-based design has been successfully applied to safety critical embedded systems [11] where behavioral properties must be guaranteed before deployment. Other successful applications of model-based design are security models [12] and patient portals [13].

Two key concepts in model-based design are *meta-modeling* [14] and *model transformations* [15], which create and relate domain specific abstractions. The semantics of model transformations has been extensively studied in the modeling community, where they are formalized as graph rewriting [16] systems, graph grammars [17], and production systems [15]. Like meta-modeling, model transformations are advantageous because they have a compact notation, formal foundation, and tool support. Model transformations are normally used for offline model synthesis; *model-driven architecture* (MDA) [18] and code synthesis [19] are two exemplars.

Modern business applications are now commonly implemented as *service-oriented architectures* (SOAs) deployed over the Web [20]. The composition of the different services has been studied in the context of work-flows, using formal techniques ranging from Petri-Nets [21] to Pi-Calculus [22].

7 Discussion and Conclusion

We presented BAM, a model-based framework for designing data-centric business applications. A system is described with three interrelated models: The data model provides an abstract characterization of the data states in which the system may find itself; high-level invariants added to the data model characterize which data states are problematic. The operation model provides an expressive framework for capturing the basic operations of the business applications. The connectivity model extends operations over arbitrary topologies while preserving the semantics of data access, information flow, and state-update. An analysis of these models yields a finite representation of the unstable states that the system might reach.

In future work, we intend to apply model checking [23] to the scenarios of I_K to determine if the operations are sufficient to correct the instabilities in the system. In fact, model checking derives the sequence of operations that evolve a system to a consistent state, and this can be used for protocol synthesis. The amount of model checking can be reduced by applying folding morphisms to the information horizons of the operations, which also produces a finite characterization of the distinct topologies touched by operations. Our final goals are **(1)** to decide if there exists a sequence of operations to stabilize any unstable system **(2)** to synthesize a protocol (sequence of operations) that stabilizes any unstable system. BAM makes this possible via a unique compositional language for specifying and analyzing this important class of systems. Finally, tools from model-based design make it possible to readily implement BAM.

References

1. Claessens, J., Preneel, B., Vandewalle, J.: A tangled world wide web of security issues. First Monday **7**(3) (2002)
2. Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and contextual integrity: Framework and applications. In: S&P. (2006) 184–198
3. Neumann, P.G.: System and network trustworthiness in perspective. In: ACM Conference on Computer and Communications Security. (2006) 1–5
4. Object Management Group: Mda guide version 1.0.1. Technical report (2003)
5. Fowler, M., Rice, D., Foemmel, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Professional (November 2002)
6. Aldred, L., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Communication abstractions for distributed business processes. In: CAiSE. (2007) 409–423
7. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3) (2001) 374–425
8. Object Management Group: Unified modeling language: Superstructure version 2.0, 3rd revised submission to omg rfp. Technical report (2003)
9. Jackson, E.K., Sztipanovits, J.: Towards a formal foundation for domain specific modeling languages. Proceedings of the Sixth ACM International Conference on Embedded Software (EMSOFT'06) (October 2006) 53–62
10. Liu, X., Liu, J., Eker, J., Lee, E.A. In: Heterogeneous Modeling and Design of Control Systems. IEEE Press and Wiley-Interscience (2003) 105–122

11. Berry, G., Kishinevsky, M., Singh, S.: System level design and verification using a synchronous language. In: ICCAD. (2003) 433–440
12. Jürjens, J., Shabalin, P.: Tools for secure systems development with uml. STTT **9**(5-6) (2007) 527–544
13. Masys, D., Baker, D., Butros, A., Cowles, K.: Giving patients access to their medical records via the internet: the pcasso experience. Journal of the American Medical Informatics Association **9**(2) (March 2002) 181–191
14. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-integrated development of embedded software. Proceedings of the IEEE **91**(1) (January 2003) 145–164
15. Csertan, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varro, D.: Viatra: Visual automated transformations for formal verification and validation of uml models. In: 17th IEEE International Conference on Automated Software Engineering. (September 2002)
16. Sprinkle, J., Agrawal, A., Levendovszky, T., Shi, F., Karsai, G.: Domain model translation using graph transformations. In: ECBS. (2003) 159–167
17. Königs, A., Schürr, A.: Multi-domain integration with mof and extended triple graph grammars. In: Language Engineering for Model-Driven Software Development. (2004)
18. Bezivin, J., Gerbé, O.: Towards a precise definition of the omg/mda framework (2001)
19. Neema, S., Kalmar, Z., Shi, F., Vizhanyo, A., Karsai, G.: A visually-specified code generator for simulink/stateflow. In: VL/HCC. (2005) 275–277
20. Breu, R., Breu, M., Hafner, M., Nowak, A.: Web service engineering - advancing a new software engineering discipline. In: ICWE. (2005) 8–18
21. Aalst, W.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
22. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for ws-bpel. Journal of Logic and Algebraic Programming **70**(1) (January 2007) 96–118
23. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. In: POPL. (1992) 342–354