From Features to Roles

Philipp Chrszon
philipp.chrszon@tu-dresden.de
Technische Universität Dresden

Christel Baier
christel.baier@tu-dresden.de
Technische Universität Dresden

Clemens Dubslaff
clemens.dubslaff@tu-dresden.de
Technische Universität Dresden

Sascha Klüppelholz
sascha.klueppelholz@tu-dresden.de
Technische Universität Dresden

ABSTRACT

The detection of interactions is a challenging task present in almost all stages of software development. In feature-oriented system design, this task is mainly investigated for interactions of features within a single system, detected by their emergent behaviors. We propose a formalism to describe interactions in hierarchies of feature-oriented systems (*hierarchical interactions*) and the actual situations where features interact (*active interplays*). Based on the observation that such interactions are also crucial in role-based systems, we introduce a compositional modeling framework based on concepts and notions of roles, comprising *role-based automata (RBAs)*. To describe RBAs, we present a modeling language that is close to the input language of the probabilistic model checker Prism. To exemplify the use of RBAs, we implemented a tool that translates RBA models into Prism and thus enables the formal analysis of functional and non-functional properties including system dynamics, contextual changes, and interactions. We carry out two case studies as a proof of concept of such analyses: First, a peer-to-peer protocol case study illustrates how undesired hierarchical interactions can be discovered automatically. Second, a case study on a self-adaptive production cell demonstrates how undesired interactions influence quality-of-service measures such as reliability and throughput.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Feature interaction*; *Model checking*.

KEYWORDS

feature-oriented systems, roles, formal methods, verification

ACM Reference Format:
Philipp Chrszon, Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. 2020. From Features to Roles. In *24th ACM International Systems and Software Product Line Conference (SPLC '20), October 19–23, 2020, MONTREAL, QC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3382025.3414962

SPLC '20, October 19–23, 2020, MONTREAL, QC, Canada

ACM ISBN 978-1-4503-7569-6/20/10…$15.00
https://doi.org/10.1145/3382025.3414962

1 INTRODUCTION

Complex systems are often built from smaller and simpler subsystems. Apart from breaking down the complexity into smaller components, another important motivation during system design phases is the reuse of components to build system variants with designated functionalities for different contexts. A very popular and successful approach to model such systems is provided through the concept of *features* [7, 8, 31] that encapsulate optional or incremental units of functionalities [47]. Most prominently, in *software product lines (SPLs)* [18] each product corresponds to a combination of features, e.g., providing home or professional versions of the system as part of the same SPL. In *dynamic feature-oriented systems* [25] features can be activated or deactivated during runtime, e.g., to upgrade or downgrade the system's functionalities. Besides the obvious application in SPLs, they provide an elegant way to describe dynamic adaptations of systems where components can be added, removed, or replaced at runtime (see, e.g., [3, 23]).

As common within most compositional approaches, features are usually not fully independent components: there are various types of dependencies between features, e.g., established via shared variables or communication actions [22]. These dependencies might lead to the well-known notion of *feature interactions* [9, 40], i.e., system behaviors that emerge due to a combination of multiple features, not deducible from their individual behaviors. On the one hand, feature interactions are desired to provide a system's functionality, but there is also the case of unintended emerging behavior that is caused by feature interactions. Detecting feature interactions is difficult since the number of possible feature combinations can be exponential in the number of features. Historically, this problem led to a crisis in the area of telecommunication systems already in the early 1980s, where undesired behaviors arose between interacting call-forwarding and call-waiting features of a telephony system [13]. Since then, many approaches have been and are still developed to address this problem also for extended notions of feature interactions. One direction is the detection of interactions in dynamic feature-oriented systems, where the temporal aspect of feature reconfigurations poses a further challenge [37]. Another instance are *quantitative feature interactions* [7], which take into account how performance measures and quality of service (QoS) change when features interact. In [43], such quantitative interactions have been investigated focusing on *higher-order interactions* that involve more than two features.

In this paper we introduce two new aspects in the field of feature interactions and investigate quantitative and functional properties of such feature interactions in both, static and dynamic settings:

(a) *hierarchical interactions* describe that behavior emerges due to multiple features in different feature-oriented systems
(b) *active interplays* describe how features actually interact at certain time points during system execution

Ad (a): In the traditional setting, feature interactions are considered within a single feature-oriented system. However, complex systems are often organized hierarchically giving rise to systems of (feature-oriented) systems. Thus, there may not only be interactions between features, but also interferences between (dynamic) feature-oriented systems which we call *hierarchical interactions*. For instance, consider a network device that can be configured either as server, relay, or client. We obtain a product line that comprises three system variants, each having exactly one active feature. As only one feature is active in each variant, feature interactions arise only between multiple network devices, e.g., when transferring a file from a server device to a client.

Ad (b): Often one is not only interested in those combinations of features that interact, but also in the sequence of events that caused the interaction and the features that were actively involved. To reason about such *active interplays*, it is necessary that the active participation of a feature in an event or action is detectable, e.g., by means of feature-annotated implementations or operational system models. This is especially important in dynamic feature-oriented systems, where interactions may depend on the temporal order of feature activations and deactivations during runtime, and where the initial feature combination may have no influence on the occurrence of the interaction.

From Features to Roles. In order to capture the aforementioned two new aspects of feature interactions, we propose to employ ideas and concepts from *role-based modeling*. The concept of roles has first been introduced in [10]. Even though roles are intuitively understood, there is no generally accepted definition [35, 44]. We take inspiration from the *Compartment Role Object Model (CROM)* [34] that unifies most of the well-established views on roles from the literature and provides a graphical notation similar to UML class diagrams [39]. *Role-based systems* comprise *naturals* to which *roles* can be bound, enabling naturals to play roles, and *compartments* as objectified contexts in which roles interact. In the network example sketched in (a), naturals correspond to single network devices, roles correspond to clients, servers, or relays, and compartments correspond to the network of devices with their roles within the network. Obviously, role-based systems can be seen as an extension of feature-oriented systems: naturals correspond to root features and roles correspond to (non-root) features. Compartments do not have a direct counterpart in feature-oriented systems. They specify the scope where the interplay between roles takes place and where roles are coordinated, constituting emergent behavior that we call *role interactions*. To this end, role interactions can be used to describe the new aspect of hierarchical interactions in systems of feature-oriented systems, showing suitability of role-based concepts to capture (a). The explicit distinction between role-binding and role-playing in role-based systems [38] makes them suitable to model and detect active interplays (b) in feature-oriented systems: role-binding corresponds to the composition features towards a system variant, while role-playing corresponds to the actual interplay of features. While the focus of conceptual modeling (e.g., using CROM) is on the component structure (similar to *feature diagrams* in the setting of feature-oriented systems [31]), operational models of role-based systems have been given little consideration in the literature (see Section 4). However, such models are key to analyze active interplays when role-based concepts shall be used to describe feature-oriented systems.

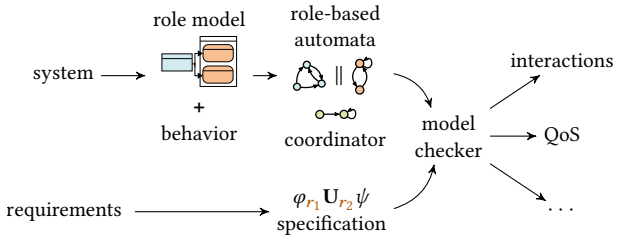


Figure 1: Overview of the analysis approach

Approach and contributions. In this paper, we propose formal modeling and analysis of role-based systems, e.g., to detect hierarchical interactions in systems of feature-oriented systems (a) and active interplays (b). Specifically, we present

(1) *role-based automata* (RBAs) and operators for role-binding and parallel composition for the representation and coordination of role-specific operational behaviors
(2) a light-weight specification language to describe role-based systems through RBAs
(3) an automated translation of our specification language to the input language of the probabilistic model checker Prism [36]
(4) case studies to illustrate how our translation can be used to analyze RBAs for detecting hierarchical interactions (a) and active interplays (b) that have an impact on functional correctness and quantitative measures

Ad (1): We specify a compositional framework by introducing *role-based automata (RBAs)* to model systems consisting of naturals, roles, and compartments, and define composition operators that realize the effect of role-binding on the structural level. Our framework is inspired by specification formalisms that describe the operational behavior of feature-oriented systems such as featured transition systems (FTSs) [17]. To this end, our composition operators also follow the concepts established in the area of feature-oriented system design, namely superimposition [22, 40] and parallel feature composition [15, 23]. An RBA is a state-based model where transitions are guarded with role annotations that describe conditions on which roles have to be actively played when taking a transition. Constraints on role-playing are captured by a component called *(role-playing) coordinator*. RBAs can also model stochastic aspects, both internal and external to the system. While internal stochastics are present, for instance, in cryptographic algorithms, as well as in coordination and communication protocols, external stochastics arise from the environment of the system, like workload or hardware defects. The semantics of an RBA w.r.t. a coordinator is defined as a *Markov decision process (MDP)* [41] that encodes role annotations into its actions and thus can be used towards a quantitative analysis of role-based behavior.

Ad (2): To specify RBAs and coordinator, we introduce a lightweight modeling language based on guarded commands [30] and reactive modules [6]. Our language is in the line of well-established guarded command languages used to describe operational behaviors of feature-oriented systems [17, 23] such as fPromela [16] and ProFeat [15]. Hence, by the aforementioned correspondences, our language can be used to specify both, feature-oriented systems and role-based systems, and eases the extension of tools for modeling and analyzing feature-oriented systems to role-based systems.

Ad (3): To exemplify how the RBA formalism can be used to analyze role-based systems and systems of feature-oriented systems, we implemented an automated translation of RBAs into the input language of the probabilistic model checker Prism [36]. The Prism language is supported by a wide range of quantitative analysis tools (see, e.g., [21, 36]). Thus, our transformation unleashes the full power of functional and quantitative analysis methods also for role-based systems modeled in the RBA specification language (2). In Figure 1, the general schema of the analysis approach is depicted.

Ad (4): We demonstrate and evaluate our analysis approach and tooling by two case studies. First, we consider a peer-to-peer file-transfer case study following the aforementioned network example. Our formal analysis reveals undesired (functional) interactions between different networks, illustrating role interaction and hierarchical interaction detection (a). Furthermore, our tooling returns role-annotated action traces as witness for the violation of certain properties, which allows us to pinpoint the situations where interactions actually take place (cf. role-playing and active interplay detection (b)). Second, we issue a production-cell case study where we focus on quantitative hierarchical interactions by evaluating reliability and throughput. With this case study, we also demonstrate how self-adaptive systems can be modeled using RBAs and role-playing coordinators and analyzed within our framework.

2 MODELING ROLE-BASED BEHAVIOR

We introduce *role-based automata* (RBAs) as uniform formalism to model naturals, roles, and compartments as essential building blocks of role-based systems. These components can be combined using *role-binding* and *parallel-composition* operators, leading to a description of the overall role-based system behavior. In essence, RBAs for modeling role-based systems have the same purpose as featured transition systems (FTSs) [17] for modeling feature-oriented systems. To this end, role-binding is similar to weaving a feature's implementation into a base system [15, 22, 23, 40], while parallel composition of RBAs corresponds to the parallel composition of isolated feature-oriented systems [17]. The main difference between our framework relying on RBAs and its feature-oriented counterparts is the capability to model compartments and the coordination of roles contained within these compartments. Note that the ability to bind roles to compartments enables a hierarchical modeling of the system, similar to the approach presented in [14] where coordination principles in role-based systems were issued. Parallel composition of RBAs formalizes the communication between RBAs using the standard notion of synchronization over shared actions. A key concept in role-based systems is the distinction between role-binding and role-playing, i.e., the *ability* to activate role's actions and actually performing role actions, respectively [38]. However, there are usually constraints on the role-playing, formalized using another automata-based component, the *(role-playing) coordinator*. While in feature-oriented systems interactions emerge from the feature implementations, the coordinator allows us to explicitly define the desired interactions between roles. The composition of an RBA with a coordinator resolves all the allowed combinations of role-playing and yields a transition system, or in the probabilistic case, a Markov decision process (MDP). As transition systems and MDPs are well-established formalisms with broad applications, our semantics enables the use of analysis tools for these formalisms.

To illustrate the role-based concepts in more detail, we use a banking example adapted from [19]. Here, the basic functionality that is provided by an account allows increasing and decreasing its balance. Transactional processing and checking for sufficient funds when transferring money is encapsulated in roles of the source and target account. We model a money transfer between two accounts Acc_1 and Acc_2, acting as source and target accounts, respectively. The transfer itself is specified by a coordinator comprising source and target roles of Acc_1 and Acc_2, respectively.

2.1 Role-based Automata

An RBA can be viewed as a featured transition system [17] that contains role annotations instead of feature annotations to formalize role-dependent behaviors. If not stated differently, sets are assumed to be finite. For a set X, we denote by 2^X the power set of X.

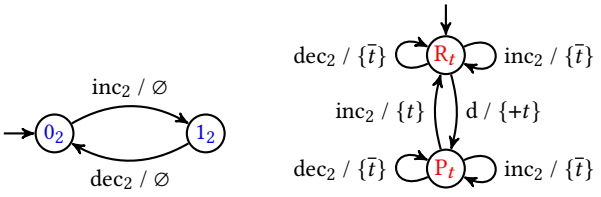
Role interfaces. We rely on *role interfaces* as pairs $R = \langle B, U\rangle$, where B and U are finite disjoint sets of instance names, standing for bound and unbound roles, respectively. We simply write R for $B \cup U$ in case we do not explicitly refer to R as a role interface. When $U = \emptyset$, the role interface is called *closed*. In case two role interfaces $\langle B_1, U_1\rangle$ and $\langle B_2, U_2\rangle$ have no common bound roles, i.e., $B_1 \cap B_2 = \emptyset$, they are called *compatible*. We define a commutative and associative composition operator $\oplus$ on compatible role interfaces where $\langle B_1, U_1\rangle \oplus \langle B_2, U_2\rangle = \langle B_1 \cup B_2, (U_1 \cup U_2) \setminus (B_1 \cup B_2)\rangle$.

Role annotations. A *role annotation* denotes which roles are played (or not played, respectively) on a transition. We define the set of role annotations as $\mathbb{A}(R) = \{r, \overline{r}, +r : r \in R\}$, where transitions annotated with r or $\overline{r}$ stand for role r is actively played, or explicitly not played, respectively. In case a transition is neither annotated with r nor $\overline{r}$, then the role r may be played, but not necessarily. Transitions annotated with $+r$ describe that an r-transition is added to the behavior of the player upon binding role r.

Definition 2.1. A *role-based automaton (RBA)* is a tuple $\mathcal{A} = (S, Act, R, \longrightarrow, S^{init})$ where S and Act are sets of states and actions, respectively, $R = \langle B, U\rangle$ is a role interface, $\longrightarrow \subseteq S \times Act \times 2^{\mathbb{A}(R)} \times S$ is a transition relation, and $S^{init} \subseteq S$ is a set of initial states.

In a transition (s, α, X, s') of an RBA, $s \in S$ is the source state, $\alpha \in Act$ is an action, $X \subseteq \mathbb{A}(R)$ is a set of role annotations, and $s' \in S$ is the target state. We assume that for all transitions and roles $r \in R$ we have that $|X \cap \{r, \overline{r}, +r\}| \leq 1$ and write $s \xrightarrow{\alpha/X} s'$ for $(s, \alpha, X, s') \in \longrightarrow$. An RBA is an *unbound role instance* of a role r if $R = \langle \emptyset, \{r\}\rangle$ and all transitions are annotated with r, $\overline{r}$, or $+r$.

Figure 2a shows the RBA for the account Acc_2. To keep the example automata small, the account balance is either 0 or 1 and can



(a) *Acc*$_2$ (b) *Target*

Figure 2: Role-based automata for an account natural and a target role with increment, decrement, and deposit actions

be increased or decreased using the *inc* and *dec* actions, respectively. Since this RBA represents a natural, all role annotations are empty. In Figure 2b, the RBA representing the target account role is shown. In the ready state (R_t), the deposit action d may be invoked, which enters the processing state (P_t). Self-loops annotated with $\{\bar{t}\}$ enable inc_2 and dec_2 actions if the *Target* role t is not played. Note that the RBA in Figure 2b is an unbound role instance of t.

2.1.1 Parallel Composition. The *parallel-composition* operator on RBAs extends the well-known definition of parallel composition on FTSs [17]. Two automata are running concurrently and communicate via handshaking, i.e., synchronization over shared actions.

Definition 2.2. Let $\mathcal{A}_i = (S_i, Act_i, R_i, \longrightarrow_i, S_i^{init})$ be two RBAs with compatible role interfaces R_i for $i \in \{1,2\}$. The *parallel composition* of $\mathcal{A}_1$ and $\mathcal{A}_2$ is defined as

$$\mathcal{A}_1 \parallel \mathcal{A}_2 \;=\; \big(S_1{\times}S_2,\ Act_1{\cup}Act_2,\ R_1{\oplus}R_2,\ \longrightarrow,\ S_1^{init}{\times}S_2^{init}\big)$$

where $\longrightarrow$ is the smallest transition relation fulfilling the rules

$$(\text{int}_1)\ \frac{s_1 \xrightarrow{\alpha/X}_1 s_1' \qquad \alpha \in Act_1 \setminus Act_2}{\langle s_1, s_2\rangle \xrightarrow{\alpha/X} \langle s_1', s_2\rangle}$$

$$(\text{int}_2)\ \frac{s_2 \xrightarrow{\alpha/Y}_2 s_2' \qquad \alpha \in Act_2 \setminus Act_1}{\langle s_1, s_2\rangle \xrightarrow{\alpha/Y} \langle s_1, s_2'\rangle}$$

$$(\text{sync})\ \frac{s_1 \xrightarrow{\alpha/X}_1 s_1' \qquad s_2 \xrightarrow{\alpha/Y}_2 s_2' \qquad \alpha \in Act_1 \cap Act_2}{\langle s_1, s_2\rangle \xrightarrow{\alpha/X\cup Y} \langle s_1', s_2'\rangle}$$

Composing two RBAs again yields an RBA. Note that due to joining the role annotations in the (sync) rule, multiple roles can be played in parallel in a single transition.

2.1.2 Role Binding. The *role-binding* operator on RBAs extends the parallel feature module composition of feature-oriented systems [15, 22, 23]. Recall that role-binding combines the behavior of a role with its player, enabling the player to play the bound role.

Definition 2.3. Let $R_a = \langle B_a, U_a\rangle$ and R_p be compatible role interfaces. *Binding* an unbound role $r \in U_a$ in $\mathcal{A} = (S_a, Act_a, R_a, \longrightarrow_a, S_a^{init})$ to a player $\mathcal{P} = (S_p, Act_p, R_p, \longrightarrow_p, S_p^{init})$ yields an RBA

$$\mathcal{A}[r \to \mathcal{P}] \;=\; \big(S_a{\times}S_p,\ Act_a{\cup}Act_p,\ R,\ \longrightarrow,\ S_a^{init}{\times}S_p^{init}\big)$$

where $R = R_a \oplus R_p \oplus \langle\{r\}, \emptyset\rangle$ and where $\longrightarrow$ is the smallest transition relation fulfilling the rules shown in Figure 4.

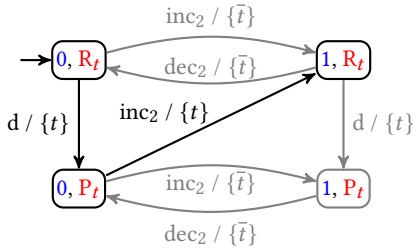


Figure 3: Result of binding *Target* to *Acc*$_2$ (*Target*$[t \to Acc_2]$)

Note that the first three rules in Figure 4 are defined as for the parallel composition (see Definition 2.2). Rule (add) covers the case where r adds and possibly overrides an action of the player to which r is bound to, without any synchronization with the player. This effectively allows the role to add new behavior to its player using the role annotation $+r$, similar to the concept of superimposition [22, 40]. For instance, the *Target* role in Figure 2b adds the action d to its player. For our running example, the result of binding the *Target* role to the player *Acc*$_2$ (see Figure 2) is shown in Figure 3. Transitions are emphasized in case role t is actively played to receive a deposit. Note that a role can also suppress transitions of the player by simply blocking them, i.e., by not providing a synchronizing transition. For instance, if the RBA for the role *Target* in Figure 2b would not provide the self-loops, then the inc_2 and dec_2 actions would not be present in the product. Changing the player's behavior can be achieved by blocking a player transition and then replacing it with a role transition. In this case, we say that the role *overrides* the player's behavior. Multiple roles can be bound to the same player by nested role binding. For instance, allowing the account *Acc*$_2$ to be the source in one transfer and the target in another can be achieved by a composition *Source*$[s \to Target[t \to Acc_2]]$.

2.1.3 Algebraic Properties of Compositions. We highlight the following algebraic properties of our composition operators:

Theorem 2.4. *For pairwise compatible RBAs $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{P}_1, \mathcal{P}_2$ where $\mathcal{A}_1, \mathcal{A}_2$ are unbound role instances with names a_1, a_2, we have*

$$\mathcal{A}_1 \parallel \mathcal{A}_2 \cong \mathcal{A}_2 \parallel \mathcal{A}_1$$
$$\mathcal{A}_1 \parallel (\mathcal{A}_2 \parallel \mathcal{A}_3) \cong (\mathcal{A}_1 \parallel \mathcal{A}_2) \parallel \mathcal{A}_3$$
$$(\mathcal{A}_1 \parallel \mathcal{A}_2)[a_1 \to \mathcal{P}_1] \cong \mathcal{A}_1[a_1 \to \mathcal{P}_1] \parallel \mathcal{A}_2$$
$$(\mathcal{A}_1 \| \mathcal{A}_2)[a_1 \to \mathcal{P}_1][a_2 \to \mathcal{P}_2] \cong (\mathcal{A}_1 \| \mathcal{A}_2)[a_2 \to \mathcal{P}_2][a_1 \to \mathcal{P}_1]$$

Here, $\cong$ stands for equivalence up to isomorphism. In case $Act_1 \cap Act_2 = \emptyset$ for the action sets Act_i of $\mathcal{A}_i$, $i \in \{1,2\}$, we further have

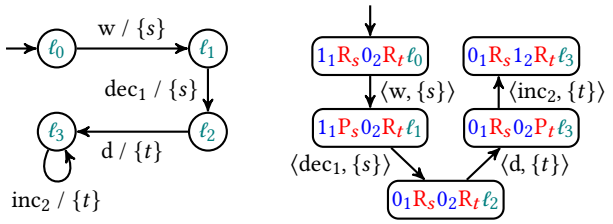
$$\mathcal{A}_1[a_1 \to \mathcal{A}_2[a_2 \to \mathcal{P}_1]] \cong \mathcal{A}_2[a_2 \to \mathcal{A}_1[a_1 \to \mathcal{P}_1]]$$

2.1.4 Non-blocking Roles. As apparent in the (sync) rule of Figure 4, a bound role can block actions of the player even if the role is not actively played, e.g., when the player is able to perform an α-transition but a bound role r does not provide an α-transition with $\bar{r}$ annotation. This phenomenon is well-known to arise also within parallel feature module composition in feature-oriented systems [15, 22]. We call a role r *non-blocking* for action α if every state s of the associated RBA contains a self-loop $(s, \alpha, \{\bar{r}\}, s)$. The role *Target* in Figure 2b provides such a self-loop for each action of

$$(\text{int}_1)\ \frac{s_a \xrightarrow{\alpha/X}_a s'_a \qquad \alpha \in Act_a \setminus Act_p}{\langle s_a, s_p\rangle \xrightarrow{\alpha/X} \langle s'_a, s_p\rangle} \qquad (\text{int}_2)\ \frac{s_p \xrightarrow{\alpha/Y}_p s'_p \qquad \alpha \in Act_p \setminus Act_a}{\langle s_a, s_p\rangle \xrightarrow{\alpha/Y} \langle s_a, s'_p\rangle}$$

$$(\text{sync})\ \frac{s_a \xrightarrow{\alpha/X}_a s'_a \qquad s_p \xrightarrow{\alpha/Y}_p s'_p \qquad \alpha \in Act_a \cap Act_p \qquad +r \notin X}{\langle s_a, s_p\rangle \xrightarrow{\alpha/X \cup Y} \langle s'_a, s'_p\rangle} \qquad (\text{add})\ \frac{s_a \xrightarrow{\alpha/X}_a s'_a \qquad \alpha \in Act_a \cap Act_p \qquad +r \in X}{\langle s_a, s_p\rangle \xrightarrow{\alpha/(X \setminus \{+r\} \cup \{r\})} \langle s'_a, s_p\rangle}$$

Figure 4: Rules for binding a role r within an RBA $\mathcal{A}$ to a player $\mathcal{P}$



(a) role-playing coordinator C (b) transition system

Figure 5: Role-playing coordinator and transition system for $[\![(\textbf{\textit{Source}}[s \to \textbf{\textit{Acc}}_1] \parallel \textbf{\textit{Target}}[t \to \textbf{\textit{Acc}}_2])]\!]_C$

player Acc_2. Thus, the original behavior of Acc_2 is preserved upon role binding (cf. horizontal transitions in Figure 3).

2.2 Coordination and Semantics of RBAs

We define a *role-playing* as a set $\mathcal{I} \subseteq R$ that contains all roles that are necessarily played, i.e., if $r \in R$ and $r \in \mathcal{I}$ the role is played and not played otherwise. Note that a role annotation in an RBA may stand for multiple role-playings provided by a set of *possible role-playings* $\mathbb{R}(X, R) \subseteq 2^R$ w.r.t. $X \in \mathbb{A}(R)$ as the set comprising exactly those $\mathcal{I} \subseteq R$ for which

(1) for all $r \in \mathcal{I}$ we have $\overline{r} \notin X$, and
(2) for all $r \in R$ with $\{r, +r\} \cap X \neq \emptyset$ we have $r \in \mathcal{I}$.

Intuitively, for any $\mathcal{I} \in \mathbb{R}(X, R)$ the rules above guarantee that $\mathcal{I}$ does not contain a role that cannot be active (1) and does contain all roles that are active (2). Role-playing in a transition $\ell \xrightarrow{\langle \alpha, X\rangle} \ell'$ of an RBA over R intuitively means that all role combinations $\mathcal{I} \in \mathbb{R}(X, R)$ could be played as they fulfill the constraints on the role-playing imposed by X.

2.2.1 Role-playing Coordinator. We specify the semantics of RBAs w.r.t. a coordination component called *(role-playing) coordinator* that specifies the rules for role-playing in an RBA. Essentially, a role-playing coordinator is an RBA but with a different semantics.

Definition 2.5. A *coordinator* is an RBA $\mathsf{C} = (S, Act, R, \longrightarrow, S^{init})$ where R is closed and where $+r \notin Y$ for all $\ell \xrightarrow{\langle \alpha, Y\rangle} \ell'$ and $r \in R$.

The coordinator formalism allows us to specify both static as well as temporal constraints on role-playing. A static constraint must hold at all times, while a temporal constraint imposes some order on the role-playing. The coordinator depicted in Figure 5a specifies, e.g., the constraint that first the *Source* role s must be played before the *Target* role t can be played. It also implicitly defines the static constraint that roles s and t can never be played simultaneously, as there is no transition annotated with $\{s, t\}$.

2.2.2 Semantics of RBAs. Given an RBA $\mathcal{A}$ and a coordinator C, the operational semantics of $\mathcal{A}$ under C is a transition system.

Definition 2.6. Composing an RBA $\mathcal{A} = (S_a, Act_a, R_a, \longrightarrow_a, S_a^{init})$ and a coordinator $\mathsf{C} = (S_c, Act_c, R_c, \longrightarrow_c, S_c^{init})$ yields

$$[\![\mathcal{A}]\!]_\mathsf{C} \;=\; \big(S_a \times S_c,\ Act,\ \longrightarrow,\ S_a^{init} \times S_c^{init}\big)$$

where $Act = (Act_a \cup Act_c) \times 2^R$ with $R = R_a \cup R_c$, and $\longrightarrow \subseteq S \times Act \times S$ is the smallest transition relation fulfilling the rules

$$(\text{int}_a)\ \frac{s \xrightarrow{\alpha/X}_a s' \qquad \alpha \in Act_a \setminus Act_c \qquad \mathcal{I} \in \mathbb{R}(X, R)}{\langle s, \ell\rangle \xrightarrow{\langle \alpha, \mathcal{I}\rangle} \langle s', \ell\rangle}$$

$$(\text{int}_c)\ \frac{\ell \xrightarrow{\alpha/Y}_c \ell' \qquad \alpha \in Act_c \setminus Act_a \qquad \mathcal{I} \in \mathbb{R}(Y, R)}{\langle s, \ell\rangle \xrightarrow{\langle \alpha, \mathcal{I}\rangle} \langle s, \ell'\rangle}$$

$$(\text{sync})\ \frac{s \xrightarrow{\alpha/X}_a s' \qquad \ell \xrightarrow{\alpha/Y}_c \ell' \qquad \alpha \in Act_a \cap Act_c \qquad \mathcal{I} \in \mathbb{R}(X \cup Y, R)}{\langle s, \ell\rangle \xrightarrow{\langle \alpha, \mathcal{I}\rangle} \langle s', \ell'\rangle}$$

The interaction between the RBA and the coordinator is formalized using synchronization over both actions and the role-playing, allowing to enforce and restrict role-playing. Additionally, the coordinator is able to react to certain role-playing, enabling the specification of temporal constraints. Clearly, for a trivial coordinator comprising only one state and an empty transition relation, a transition-system semantics for RBAs is provided by resolving every possible role-playing nondeterministically due to (int_a) in being the only rule applicable. Figure 5b shows the transition system for the banking example with two accounts, one playing the *Source* role and the other playing the *Target* role, under the coordinator shown in 5a. The coordinator realizes that money is withdrawn from the source account by decrementing its balance and then deposit money on the target account.

The transition systems as underlying semantics for role-based systems allow us to apply methods for simulation and verification, such as standard model-checking algorithms for temporal logics such as CTL and LTL [12]. To reason about role-playing, which is encoded into the actions of the resulting transition system, standard approaches using action-based logics may be employed [20].

2.3 Probabilistic Extension

We presented a non-probabilistic framework to emphasize the main concepts that, however, can be also extended to include probabilistic choice. Such an extension enables reasoning about probabilities of temporal events and other quantitative measures, e.g., expected energy consumption. To this end, we modify the transition relation

```
natural type Account;
role type Source(Account); role type Target(Account);
compartment type MoneyTransfer(Source, Target);
```

Listing 6: Role model for the banking example

```
impl Account {
  bal : [0..MAX_BAL] init init_balance[index(self)];
  [self.inc] bal < MAX_BAL -> (bal' = bal + 1);
  [self.dec] bal > 0 -> (bal' = bal - 1);
  [self.deposit] false -> true;
  [self.withdraw] false -> true; }

impl Target {
  s : enum { READY, PROCESSING } init READY;
  [override player.deposit] s = READY -> (s' = PROCESSING);
  [player.inc] s = PROCESSING -> (s' = READY); }
```

Listing 7: Behavior for the `Account` and `Target` types

of RBAs as shown in Definition 2.1 to include a probabilistic choice of the successor state, formally $\longrightarrow \subseteq S \times Act \times 2^{\mathbb{A}(R)} \times Distr(S)$ where $Distr(S)$ denotes the set of all probability distributions over S. The necessary changes to the composition operators and the definition of the role-playing coordinator are straightforward. We refer to the supplementary material [2] for the modified definitions. With these modifications in place, the semantics of an RBA under a role-playing coordinator is no longer a transition system, but becomes a *Markov decision process (MDP)* [41].

2.4 Role-based Modeling Language

To specify (probabilistic) role-based systems through RBAs, we developed a role-based modeling language that comprises a role model and behavioral component specifications. The distinction between these two descriptions is inspired by modeling languages for feature-oriented systems (see, e.g., [15, 16, 22]), where the system is described through a feature model and behavioral specifications for feature modules. While the role-model part in our language is based on existing languages commonly used in conceptual modeling [34], RBAs defining component behaviors are given through a guarded command language [6, 30] that is similar to the one used to describe models for the probabilistic model checker Prism.

A role model comprises a set of natural types, role types, and compartment types. Listing 6 shows the role model for the banking example. A role-type definition includes a list of possible player types, e.g., a role of type `Source` can be played by a natural of type `Account`. A compartment-type definition includes a list of role types, meaning that an instance of the compartment must contain a role for each of the listed role types.

The behavior of a component is defined by an `impl` block. It may contain one or more variables that define the state space of the component, followed by a list of guarded commands describing the transitions between states. A guarded command has the form `[action] guard -> update`, where `guard` is a Boolean expression over constraints on variables. If the `guard` expression evaluates to `true` in some variable evaluation, the component can execute the command and update its local variables. If an `action` is given, the

```
system {
  acc[2] : Account; src : Source; trg : Target;
  src boundto acc[0]; trg boundto acc[1];
  mt : MoneyTransfer; src in mt; trg in mt; }
```

Listing 8: Instantiation of component types, role binding, and assigning roles to compartments

```
coordinator {
  l : [0..3] init 0;
  [player(src).withdraw] [src]  l = 0 -> (l' = 1);
  [player(src).dec] [src]       l = 1 -> (l' = 2);
  [player(trg).deposit] [trg]   l = 2 -> (l' = 3);
  [player(trg).inc] [trg]       l = 3 -> (l' = 3); }
```

Listing 9: Role-playing coordinator for a money transfer

command will synchronize with commands of other components that are labeled with the same action. Listing 7 shows the behavior definition of the `Account` natural and the `Target` role. Stochastic updates are defined by specifying a probability distribution over variable updates. For instance, to express that increasing the balance of an account may fail with a probability of 0.01, the following command may be used:

```
[self.inc] bal < MAX_BAL -> 0.99:(bal' = bal+1) + 0.01:true;
```

Note that an implementation is provided for types rather than for instances. Creating multiple components with similar behavior can be achieved by instantiating a component type. The `self` keyword then stands for the concrete instance name upon instantiation. Similarly, the `player` keyword refers to the player instance that the role is bound to. The commands in lines 5 and 6 of Listing 7 state that an instance of `Account` by itself will never execute the `deposit` and `withdraw` actions, signified by the guard `false`. If an action is preceded by `override` (line 10), this action will not synchronize with the player. Instead, the action of the role replaces the action of the player. Thus, an `Account` instance can actually execute the `deposit` action, but only if it plays a `Target` role. Formally, all transitions defined by an `override` command carry the role annotation $\{+r\}$ where r is the role instance name. For `MAX_BAL=1`, one instantiation of Listing 7 represents the RBAs shown in Figure 2.

A system instance is specified using a `system` block as shown in Listing 8. Instances of component types are created by providing a name followed by a type (lines 2–4). In line 2, an array of two `Account` instances is defined. Binding a role instance to a player is achieved by the `boundto` keyword. Similarly, the `in` keyword defines the membership of a role in a compartment.

Coordinator commands have the form `[action] [role-guard] guard -> update`, where `role-guard` is a Boolean expression over role instances defined in the model (see Listing 9). A coordinator command synchronizes with exactly those transitions of the system, whose role annotation satisfies the role guard. For instance, line 4 specifies that the player of the `src` role (`acc[0]`) may execute the withdraw action, but only if it plays the `src` role. The definition in Listing 9 yields the coordinator shown in Figure 5a.

To reason about quantitative properties such as energy consumption and throughput, states, transitions, and role playing can be annotated with costs and rewards.

3 EVALUATION

To exemplify the use of our RBA framework for analyzing role-based systems, e.g., to detect hierarchical interactions and to identify active interplays, we implemented a tool that translates systems given in the specification language presented in Section 2.4 into the input language of the probabilistic model checker Prism [36]. We present two illustrative examples where the actual analysis is performed by Prism on automatically translated role-based models. The first example relates to a simple peer-to-peer file transfer protocol where the focus is to find and eliminate interactions that cause undesired side-effects when connecting two networks. This example also demonstrates the scalability of our approach. The second example deals with a self-adaptive robot production cell, focusing on quantitative effects of feature interactions. The implementation and the experiments are provided in the corresponding artifact [1] for this paper[1].

3.1 Translation into Prism

According to the theoretical framework of Section 2, a (probabilistic) role-based system is constructed by composing the RBAs for the naturals, roles, and compartments, followed by a composition with the coordinator component towards a single MDP. A translation to Prism's input language could hence be conducted by explicitly encoding the resulting MDP, possibly suffering from an exponential blowup of the model description. We hence apply an alternative method that preserves the modular structure of RBAs with a suitable compositional structure in Prism. First, we translate each RBA $\mathcal{A}$ into a separate MDP module $\mathcal{M}[\mathcal{A}]$ with multi-actions[2] [11] where possible role-playings are encoded in the transitions as part of the action label[3]. The resulting multi-action MDPs are then combined by parallel composition [11]. For binding a role r in an RBA $\mathcal{R}$ to some player, we define a closure operation $cl^r_{\mathcal{R}}(\mathcal{M})$ on multi-action MDPs $\mathcal{M}$ that adds self-loops to all states/actions that have a $+r$-transition of $\mathcal{R}$. Thus, an overriding transition of $\mathcal{R}$ is not blocked by the player and the player remains in the same state. Correctness of this approach is provided by the following theorem.

Theorem 3.1. *Let $\mathcal{A}$ and $\mathcal{B}$ be RBAs with compatible role interfaces, $\mathcal{R}$ be an RBA for role r that can be bound to $\mathcal{A}$ and $\cong$ denote equivalence up to isomorphism. Then,*

(1) $\mathcal{M}\big[\mathcal{A} \| \mathcal{B}\big] \;\cong\; \mathcal{M}[\mathcal{A}] \parallel \mathcal{M}[\mathcal{B}]$
(2) $\mathcal{M}\big[\mathcal{R}[r \to \mathcal{A}]\big] \;\cong\; cl^r_{\mathcal{R}}(\mathcal{M}[\mathcal{A}]) \parallel \mathcal{M}[\mathcal{R}]$

In an optional third step of the translation sketched above, the resulting Prism model can be transformed such that the role-playings are encoded into state labels [20]. This allows for expressing and analyzing role-playing properties with standard Prism property specifications. There is also a heuristics for the translation of the coordinator that might avoid the exponential blowup of Prism code lines in the number of roles. Naively translated into a single Prism module, the coordinator module would contain all roles in its action alphabet. Thus, translating a single command would lead to one command for each satisfying evaluation of the role guard (cf. definition for possible role-playings in Section 2.2). In case the role-guards of different commands are independent, we propose a more efficient translation with separate coordinator modules realizing local control for role-playing whenever possible. For this, we first construct a graph over coordinator commands where two nodes are connected if the sets of coordinated roles overlap. The connected components then induce a partitioning of the commands. Second, a module for each partition is generated. Each of these modules has a smaller action alphabet than the monolithic coordinator module, resulting in a reduced number of commands. This optimization proved to be crucial for the analysis of our first example as it reduced the number of commands in the translated Prism models significantly.

3.2 Peer-to-peer File Transfer

Our first example issues a peer-to-peer file transfer scenario, inspired from [27] and already mentioned in the introduction to motivate hierarchical interactions. The system comprises a number of computer systems called *stations* that constitute the naturals. Each station can store one or more files and play the roles of a *client*, *server*, and *relay* to send requests for files, provide files to other stations, and relay files to connected peers within the network, respectively. Note that up to this stage, the system can be interpreted as a feature-oriented system where roles correspond to features of the network devices. The role-specific properties of the model come into play when a *network* compartment is considered that defines the topological structure of the network and specifies the peer-to-peer protocol, i.e., specifying inter-system coordination. Stations can be shared among multiple network compartments, while role instances belong to exactly one network. For example, a station can be part of two networks and play server roles from both network compartments concurrently. A file transfer is initiated by a client sending a request message. If another station owns a copy of the requested file, it then acts as a server for this file. A server fetches the file from its station and sends the requested file back to the client, possibly via some relays. The client then stores the file on its station. The described file-transfer protocol is implemented in terms of a coordinator. In addition to this, each network prevents overwriting the last remaining copy of a file in the network, such that no files are lost. For simplicity, we assume that within one network only a single file transfer can happen at any point. The model is configurable and can be instantiated for different numbers of files, different topologies, e.g., star and ring, as well as different storage capacities of the stations. The model is specified using the role-based input language defined in Section 2.4 and then translated into the standard Prism language (see Section 3.1 for details). The underlying semantics is hence provided by an MDP with nondeterministic choice among the clients for producing the next request. This MDP yields the basis for the following analysis.

3.2.1 Functional Analysis. We first establish the functional correctness of the system and the model for the case of a single network. To track the progress of the system, we added an additional monitoring component containing Boolean variables f_i^s for all stations s and all files i, which are all initially set to *false*. If a station s receives a file i, the monitor sets f_i^s to *true*. We further define the atomic propositions $send_c$ and $recv_c$ that hold in all states where the client

[1] For the current version of the implementation, see https://github.com/pchrszon/rbsc
[2] In multi-action MDPs transitions are labeled by action sets instead of single actions.
[3] Formalization of the transformation is provided in the supplementary material [2].

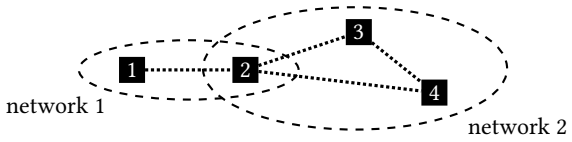


Figure 10: Scenario with two overlapping networks

c has just sent a request and received the requested file, respectively. In addition to checking for the absence of deadlocks, we checked the following properties given as CTL formulas. Here, S denotes the set of stations, C the set of client roles, and I the set of files:

(1) Some station is able to get file A: $\forall\Box\exists\Diamond\big(\bigvee_{s\in S} f_A^s\big)$
(2) All stations can eventually receive each file at least once: $\exists\Diamond\big(\bigwedge_{s\in S, i\in I} f_i^s\big)$
(3) Each request will eventually be answered: $\bigwedge_{c\in C}\forall\Box\big(send_c \implies \forall\Diamond recv_c\big)$

Using the capabilities of Prism to verify functional properties, we could show that all the above formulas are satisfied for systems that consist of a single isolated network[4].

Hierarchical interactions. In systems of connected networks sharing one or more stations, however, properties (1–3) are *not* fulfilled anymore. Consider the topology shown in Figure 10. There are two network compartments, one containing the stations 1 and 2, and one spanning the stations 2, 3, and 4. Note that since station 2 is part of both networks, it also has two sets of server, client, and relay roles bound to it such that it can play these roles in either network. We further assume that stations 2 and 3 initially store file B and station 4 stores file A. Checking the properties (1–3) on this model using Prism revealed the following unforeseen hierarchical interactions. Since the two networks may process file transfers concurrently, it is possible to lose the last remaining copy of a file: first, the client role of station 2 in network 2 requests and receives file A, but does not immediately store it on station 2. Likewise, the other client role of station 2 requests and receives file B in network 1 and also does not store it immediately. Then, file A is stored on station 2. This now allows station 4 to receive file B and to overwrite its copy of file A. Finally, the client role of station 2 stores file B, thereby overwriting the last remaining copy of A. From this point on, property (1) can no longer be satisfied. Similarly, property (3) may be violated due to concurrent file transfers: Suppose station 1 decides to send a request for file B. Before actually sending the request, station 2 requests and receives file A in network 2, thereby overwriting file B on station 2. Then, station 1 actually sends its request to station 2, which will now act as relay, since it does not have the requested file. The request is then sent back to station 1, which will also act as relay, repeating the process indefinitely.

Active interplays. While checking the above properties, Prism reports on counterexamples to witness their violation. Listing 11 presents the action trace[5] for the violation of property (3). Each line corresponds to a single system state that has been reached by executing the given action while the listed roles were played

[4]Property (3) required a fair scheduling among the different stations.
[5]For better readability, the states have been omitted and the trace has been reformatted, but has not been altered otherwise.

```
role-playing                                    action
client_0,                                       client_0_genreq_2
client_2,                                       client_2_genreq_1
client_2, not_relay_2, not_server_3, relay_3,   r_2_3_1
not_client_3, not_relay_4, relay_3, server_4,   r_3_4_1
server_4,                                       station_3_load_1
not_client_3, not_relay_4, relay_3, server_4,   d_4_3_1
client_2, not_relay_2, not_server_3, relay_3,   d_3_2_1
client_1, client_2,                             station_1_store_1
client_0, not_relay_0, not_server_1, relay_1,   r_0_1_2
not_client_1, not_server_0, relay_0, relay_1,   r_1_0_2
not_client_0, not_server_1, relay_0, relay_1,   r_0_1_2
not_client_1, not_server_0, relay_0, relay_1,   r_1_0_2
```

Listing 11: Action trace for the violation of property (3)

(or explicitly not played, indicated by the prefix "not_"). Since all role-playings are encoded into the actions of the MDP semantics of the system (cf. Definition 2.6), those roles actively involved in interactions can be easily traced down and could help to resolve undesired behavior.

3.2.2 Interactions in Feature-oriented Systems. Since roles can be interpreted as features of single network devices, the aforementioned analysis approach is applicable to detect hierarchical interactions and active interplays in systems of feature-oriented systems. In our file-transfer example, the results of the functional analysis revealed that network-local coordination alone is not sufficient in case of interacting networks with shared stations. Fixing (undesired) hierarchical interactions might hence require coordination within multiple networks, i.e., systems of feature-oriented systems. Here, the role-based view on feature-oriented systems is beneficial and provides coordination capabilities also between multiple systems through the concept of compartments, not immediate in feature-oriented systems. Using these concepts we could trace down the reasons of undesired hierarchical interactions by examining active interplays (cf. Listing 11) and then extend the coordinator by additional global constraints to prevent undesired interactions.

3.2.3 Scalability. We investigated the scalability of our analysis approach by instantiating the file-transfer model with a star topology for increasing numbers of stations and files. In order to determine the impact of the role-based modeling approach on the analysis performance, we created a second model for one network, solely relying on standard Prism language capabilities, i.e., containing neither role binding nor coordinator. Note that the standard Prism model is tailored to the concrete benchmark scenario and is less modular and flexible than the role-based model. For instance, it would require extensive rewriting of the existing model to add a second network. Furthermore, the standard Prism model does not contain any annotations, such that active interplays are not exposed which makes the detection of interactions much harder. Table 1 shows the model sizes for a given number of stations and files. The size is listed both in terms of reachable states and the number of nodes in the multi-terminal binary decision diagram (MTBDD) that Prism uses to symbolically represent the MDP. The number of nodes has been minimized for all instances via reordering [33]. The time for checking the functional properties (1–3) has been averaged over three runs with an additional warm-up run beforehand. The time for translating the role-based models is not

Table 1: Model sizes, build times, and analysis times for selected file-transfer models

			role-based model			standard PRISM model		
stations	files	states	nodes	build (s)	analysis (s)	nodes	build (s)	analysis (s)
2	1	20	2 401	0.114	0.016	384	0.047	0.017
3	2	981	45 127	1.368	0.129	5 533	0.221	0.150
9	1	58 007	185 849	10.658	1.172	34 362	1.581	1.628
4	3	235 843	208 195	9.494	1.525	23 171	2.230	1.612
5	4	193 690 961	683 121	93.195	86.554	123 723	66.381	97.930
9	2	326 024 459	884 738	133.810	83.126	411 685	147.653	115.856
7	3	4 722 943 030	1 114 291	544.446	852.297	793 286	1527.186	1911.909

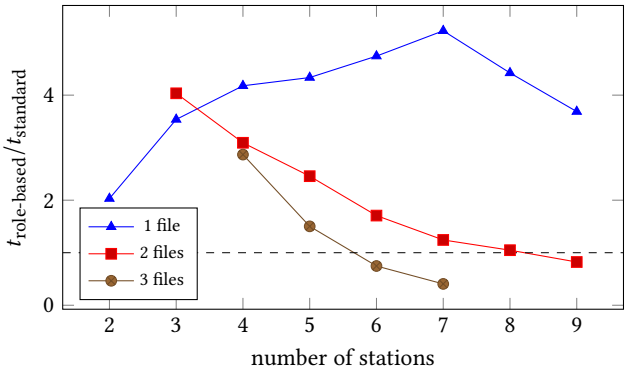


Figure 12: Combined build and analysis time of role-based model relative to the standard model for fixed numbers of files

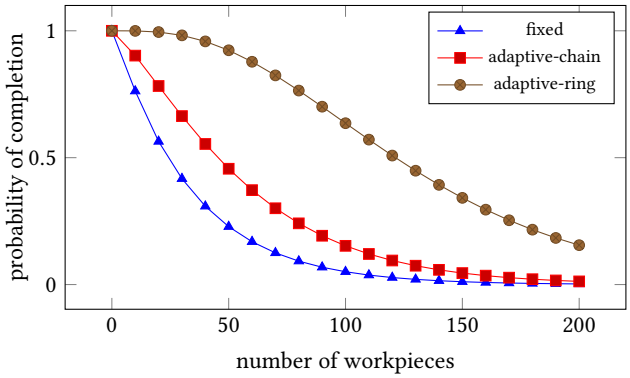


Figure 13: Maximal probability of processing n workpieces

included in the build time, since it accounted for less than 1% of the overall analysis time (for the largest instance, translating took 1.2s on average). All experiments have been carried out on a system with two quad-core Intel Xeon L5630 CPUs (at 2.13 GHz) and 192 GB RAM running Debian 10. Figure 12 visualizes the analysis time of the role-based model in relation to the standard Prism model. The overhead of the role-oriented approach is caused by the additional role-playing actions present in the MDP that also have to be encoded in the MTBDD. Furthermore, the multi-action extension of Prism [11] uses a different encoding of actions compared to standard Prism which causes an additional overhead. Nevertheless, for larger instances the impact of this overhead diminishes. For the largest instances with 2 or 3 files, the structure of the role-based model turned out to be favorable and lead to a faster analysis. In conclusion, the experiments showed that the role-oriented analysis approach is feasible even for large models.

3.3 Self-adaptive Production Cell

As a second example, we consider an automated production cell that adapts itself in case of failures (inspired by [26]). A production cell consists of multiple robots and carts that transport workpieces between robots. Each robot is equipped with a tool to fulfill a certain task. For instance, in a production cell with three robots, the first drills a hole, the second inserts a screw, and the third one tightens it. Each robot is able to perform all tasks by switching its tool. However, it is assumed that switching tools takes a considerable amount of time. Every time a tool is used, it may break with a given probability. If a workpiece cannot be processed further because of a broken tool, the cell is reconfigured locally by switching tools of the robots to restore its capability to process more workpieces. Furthermore, global reconfiguration can take place to adapt other production cells at the same time. Similar to [26] we applied a role-based approach to model the production cells. The role of a robot determines its assigned tool, e.g., a robot playing the *driller* role drills holes. We checked the functional correctness of the model before applying any quantitative analysis.

First, we analyzed the benefit of self-adaptation mechanisms w.r.t. resilience. For this, we considered three variants of a production cell with three robots. The *fixed* variant possesses no self-adaptivity and provides a baseline, while the other two are self-adaptive. In the *adaptive-chain* variant, robots are aligned in a row and only the direction in which the workpieces move through the cell can be changed. The *adaptive-ring* variant allows transporting workpieces between any two robots in any direction, thus each robot can potentially perform any task. We determined the maximal probability to process n workpieces before the cell becomes inoperable. For a probability of 0.01 that a tool breaks upon usage, the results are shown in Figure 13. Compared to the *fixed* variant, adding adaptivity significantly increases the system's resilience.

In another scenario, we assume that one of the robots can process workpieces twice as fast as the others. To fully utilize this robot, we might share it between two production cells to increase the throughput of production as shown in Figure 14. We define the throughput as the number of finished workpieces per time unit where each processing step and each reconfiguration of a production cell takes one time unit. Note that robots can work in parallel, e.g., robots

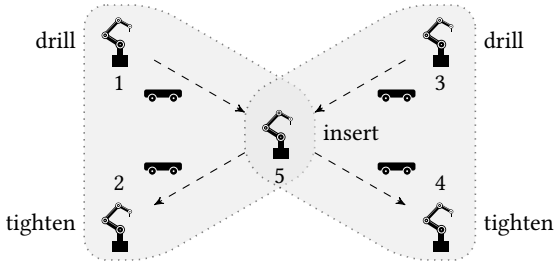


Figure 14: Two automated production cells sharing a robot

Table 2: Expected throughput of production cells

production cells	adaptation	expected throughput min	max
1	localized	0.2531	0.2827
2 (shared robot)	localized	0.3493	0.5096
2 (shared robot)	global	0.4035	0.5474
2	localized	0.6870	0.7964

1 and 3 may drill two different workpieces within one time unit. From now on, assume that the probability of breaking a tool is 0.1. We compared a single production cell with three robots with the system shown in Figure 14. The results are presented in the first two lines of Table 2. As reference, the throughput of two isolated production cells is shown in the last line. In both, best and worst case, the shared robot variant has a significantly higher throughput than a single cell while only requiring two additional robots. Note that the throughput of two isolated cells is more than double that of a single cell because of the additional redundancy of the system which allows producing more workpieces until complete failure.

While adding a second overlapping production cell increases the overall throughput, the shared robot can lead to hierarchical interactions. The shared robot cannot use two different tools at the same time, hence the choice of the assigned tool influences both production cells. Suppose that the *drill* tool of robot 1 breaks (cf. Figure 14). To process further workpieces, the left cell adapts itself by swapping the tools of robot 1 and 5. But then, there is no longer any robot equipped with the *insert* tool in the right cell. Thus, the next time a screw needs to be inserted, the right cell might reassign the *insert* tool to the shared robot 5. However, the left cell will adapt again once a hole needs to be drilled, requiring an adaption in the right cell later, and so on. Note that the interaction of frequent conflicting adaptations does not influence to functional correctness of the system as both production cells can still fully process workpieces. However, since reconfigurations take time, such frequent adaptations decrease the throughput of the overall system. This interaction can be mitigated by using a global adaption scheme that takes all production cells into account. When the *drill* tool of robot 1 breaks, the *driller* role can be assigned to robot 5 and the *inserter* role to both robot 1 and 3. This adaptation restores the processing capability of the left cell and also avoids the frequent reconfiguration described above. To quantify the impact of the interaction, we compared the expected throughput of a model with a global adaptation mechanism with local adaptations only. The results are shown in the second and third line of Table 2. Even though a global adaptation scheme requires a more sophisticated reconfiguration mechanism, it increases the throughput of the system.

4 DISCUSSION AND CONCLUSION

Towards detecting hierarchical interactions and active interplays, we presented a compositional modeling approach that relies on concepts from role-based modeling. We proposed RBAs to model the operational behavior of naturals, roles, and compartments and introduced a light-weight modeling language to describe RBAs. We implemented an automated translation from our modeling language into the input language of the probabilistic model checker Prism, which allowed us to perform formal analysis to detect and quantify hierarchical interactions and active interplays using Prism.

Further related work. We here focus on work regarding formal analysis techniques for systems that incorporate roles. While there are existing formal approaches to specify and validate role-based systems on the conceptual level [34], support for analyzing their behavior is rather limited. Hennicker and Klarl proposed the Helena approach [27] where roles played by components collaborate in ensembles towards fulfilling a common goal. In HelenaLight, components are passive as they only provide data and operations invoked by their roles. This limited interaction between roles and their players is fully covered by our approach. The operational semantics of HelenaLight [28, 32] is defined in terms of transition systems that can be verified using the model checker Spin [29]. A modular model-checking approach for collaboration-based systems was presented in [24] and extended in [45, 46]. Here, role-binding is realized by taking the union of the transition systems of role and player, and subsequently introducing additional transitions between them for switching between their behaviors. Roles have also been considered in architecture description languages, e.g., in [5]. Here, roles define the obligations and expected behaviors of components rather than behavioral adaptations. None of the aforementioned approaches deal with stochastic behaviors. Several analysis approaches for *role-based access control* policies have been presented in the literature, e.g., [4, 42, 48]. In this context, roles are mainly associated with rights and bring no operational behavior themselves.

Future work. Several directions for extending our approach are left for future work. While the concept of compartments proved to be useful in our case studies, we did not fully demonstrate their potential, e.g., for models with a deeper hierarchical structure as sketched in [14]. Other interesting directions are the development of a graphical notation for RBAs similar to UML statecharts [39], explicit formal support for role-specific requirements, and the integration of our tool with existing modeling tools.

ACKNOWLEDGMENTS

The authors are supported by the DFG through the Collaborative Research Centers CRC 912 (HAEC) and TRR 248 (see https://perspicuous-computing.science, project ID 389792660), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy, and the Research Training Groups QuantLA (GRK 1763) and RoSI (GRK 1907).

REFERENCES

[1] 2020. From Features to Roles: Artifact. https://doi.org/10.5281/zenodo.4048533

[2] 2020. From Features to Roles: supplementary material. https://wwwtcs.inf.tu-dresden.de/ALGI/TR/SPLC20

[3] Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, and Jean-Paul Rigault. 2009. Modeling Context and Dynamic Adaptations with Feature Models. In *4th International Workshop Models@run.time at Models 2009 (MRT'09)*. 10.

[4] Tanvir Ahmed and Anand R Tripathi. 2003. Static verification of security requirements in role based CSCW systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*. ACM, 196–203.

[5] Robert Allen and David Garlan. 1997. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol.* 6, 3 (1997), 213–249. https://doi.org/10.1145/258077.258078

[6] R. Alur and T. A. Henzinger. 1999. Reactive Modules. *Formal Methods in System Design* 15, 1 (1999), 7–48. https://doi.org/10.1023/A:1008739929481

[7] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer.

[8] Sven Apel and Christian Kästner. 2009. An Overview of Feature-Oriented Software Development. *Journal of Object Technology* 8 (2009), 49–84.

[9] Sven Apel, Alexander von Rhein, Thomas Thüm, and Christian Kästner. 2013. Feature-interaction detection based on feature-based specifications. *Computer Networks* 57, 12 (2013), 2399–2409. https://doi.org/10.1016/j.comnet.2013.02.025

[10] Charles W. Bachman and Manilal Daya. 1977. The Role Concept in Data Models. In *Proceedings of the Third International Conference on Very Large Data Bases - Volume 3* (Tokyo, Japan) *(VLDB '77)*. VLDB Endowment, 464–476. http://dl.acm.org/citation.cfm?id=1286580.1286629

[11] Christel Baier, Philipp Chrszon, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. 2018. Energy-Utility Analysis of Probabilistic Systems with Exogenous Coordination. In *It's All About Coordination - Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab*. 38–56.

[12] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press.

[13] Muffy Calder, Mario Kolberg, Evan H. Magill, and Stephan Reiff-Marganiec. 2003. Feature interaction: a critical review and considered forecast. *Computer Networks* 41, 1 (2003), 115–141. https://doi.org/10.1016/S1389-1286(02)00352-3

[14] Philipp Chrszon, Clemens Dubslaff, Christel Baier, Joachim Klein, and Sascha Klüppelholz. 2016. Modeling Role-Based Systems with Exogenous Coordination. In *Theory and Practice of Formal Methods (LNCS, Vol. 9660)*. Springer, 122–139.

[15] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. 2018. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects of Computing* 30, 1 (2018), 45–75.

[16] Andreas Classen, Maxime Cordy, Patrick Heymans, Axel Legay, and Pierre-Yves Schobbens. 2012. Model checking software product lines with SNIP. *International Journal on Software Tools for Technology Transfer* 14, 5 (01 Oct 2012), 589–612. https://doi.org/10.1007/s10009-012-0234-1

[17] Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-François Raskin. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Transactions on Software Engineering* 39, 8 (2013), 1069–1089.

[18] Paul Clements and Linda Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional.

[19] James O. Coplien and Trygve Reenskaug. 2014. The DCI Paradigm: Taking Object Orientation into the Architecture World. In *Agile Software Architecture*, Muhammad Ali Babar, Alan W. Brown, and Ivan Mistrik (Eds.). Morgan Kaufmann, Boston, 25 – 62. https://doi.org/10.1016/B978-0-12-407772-0.00002-2

[20] Rocco De Nicola and Frits Vaandrager. 1990. Action versus state based logics for transition systems. In *Proceedings of LITP Spring School on Theoretical Computer Science 1990 (LNCS, Vol. 469)*, Irène Guessarian (Ed.). Springer, Berlin, Heidelberg, 407–419.

[21] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. 2017. A Storm is Coming: A Modern Probabilistic Model Checker. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10427)*, Rupak Majumdar and Viktor Kuncak (Eds.). Springer, 592–600. https://doi.org/10.1007/978-3-319-63390-9_31

[22] Clemens Dubslaff. 2019. Compositional Feature-Oriented Systems. In *Software Engineering and Formal Methods*, Peter Csaba Ölveczky and Gwen Salaün (Eds.). Springer International Publishing, Cham, 162–180.

[23] Clemens Dubslaff, Christel Baier, and Sascha Klüppelholz. 2015. Probabilistic Model Checking for Feature-Oriented Systems. *Trans. Aspect-Oriented Software Development* 12 (2015), 180–220.

[24] Kathi Fisler and Shriram Krishnamurthi. 2001. Modular verification of collaboration-based software designs. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*. 152–163.

[25] Hassan Gomaa and Mohamed Hussein. 2003. Dynamic Software Reconfiguration in Software Product Families. In *PFE*. 435–444.

[26] Matthias Güdemann, Frank Ortmeier, and Wolfgang Reif. 2006. Formal Modeling and Verification of Systems with Self-x Properties. In *Autonomic and Trusted Computing, Third International Conference, ATC 2006, Wuhan, China, September 3-6, 2006, Proceedings*. 38–47. https://doi.org/10.1007/11839569_4

[27] Rolf Hennicker and Annabelle Klarl. 2014. Foundations for Ensemble Modeling – The Helena Approach. In *Specification, Algebra, and Software*, Shusaku Iida, José Meseguer, and Kazuhiro Ogata (Eds.). Lecture Notes in Computer Science, Vol. 8373. Springer Berlin Heidelberg, 359–381.

[28] Rolf Hennicker, Annabelle Klarl, and Martin Wirsing. 2015. Model-Checking Helena Ensembles with SPIN. In *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*. 331–360.

[29] Gerard J Holzmann. 2004. *The SPIN model checker: Primer and reference manual*. Vol. 1003. Addison-Wesley Reading.

[30] He Jifeng, K. Seidel, and A. McIver. 1997. Probabilistic models for the guarded command language. *Science of Computer Programming* 28, 2 (1997), 171 – 192. Formal Specifications: Foundations, Methods, Tools and Applications.

[31] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report. Carnegie-Mellon University Software Engineering Institute.

[32] Annabelle Klarl. 2015. From Helena Ensemble Specifications to Promela Verification Models. In *Model Checking Software - 22nd International Symposium, SPIN 2015, Stellenbosch, South Africa, August 24-26, 2015, Proceedings*. 39–45.

[33] Joachim Klein, Christel Baier, Philipp Chrszon, Marcus Daum, Clemens Dubslaff, Sascha Klüppelholz, Steffen Märcker, and David Müller. 2018. Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata. *STTT* 20, 2 (2018), 179–194.

[34] Thomas Kühn, Stephan Böhme, Sebastian Götz, and Uwe Aßmann. 2015. A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, SLE 2015, Pittsburgh, PA, USA, October 25-27, 2015*. 113–124. https://doi.org/10.1145/2814251.2814255

[35] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. 2014. A Metamodel Family for Role-Based Modeling and Programming Languages. In *Software Language Engineering*. Springer, 141–160.

[36] Marta Kwiatkowska, Gethin Norman, and David Parker. 2002. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer, 200–204.

[37] Y. Liu and R. Meier. 2009. Resource-Aware Contracts for Addressing Feature Interaction in Dynamic Adaptive Systems. In *2009 Fifth International Conference on Autonomic and Autonomous Systems*. 346–350.

[38] Riichiro Mizoguchi, Kouji Kozaki, and Yoshinobu Kitamura. 2012. Ontological analyses of roles. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*. IEEE, 489–496.

[39] Object Management Group OMG. 2011. Unified Modeling Language (UML): Superstructure Version 2.4.1. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/.

[40] Malte Plath and Mark Ryan. 2001. Feature integration using a feature construct. *Science of Computer Programming* 41, 1 (2001), 53–84.

[41] M. L. Puterman. 1994. *Markov Decision Processes*. Wiley. https://doi.org/10.1002/9780470316887

[42] Hind Rakkay and Hanifa Boucheneb. 2009. Security Analysis of Role Based Access Control Models Using Colored Petri Nets and CPNtools. In *Transactions on Computational Science IV*. Lecture Notes in Computer Science, Vol. 5430. Springer Berlin Heidelberg, 149–176. https://doi.org/10.1007/978-3-642-01004-0_9

[43] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don S. Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. 167–177.

[44] Friedrich Steimann. 2000. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering* 35, 1 (2000), 83–106.

[45] Nguyen Truong Thang and Takuya Katayama. 2003. Dynamic Behavior and Protocol Models for Incremental Changes among a Set of Collaborative Objects. In *6th International Workshop on Principles of Software Evolution (IWPSE 2003), 1-2 September 2003, Helsinki, Finland*. 45–50.

[46] Nguyen Truong Thang and Takuya Katayama. 2003. Towards a Sound Modular Model Checking of Collaboration-Based Software Designs. In *10th Asia-Pacific Software Engineering Conference (APSEC 2003), Chiang Mai, Thailand*. 88–97.

[47] Pamela Zave. 2001. Feature-Oriented Description, Formal Methods, and DFC. In *Language Constructs for Describing Features*, Stephen Gilmore and Mark Ryan (Eds.). Springer London, London, 11–26.

[48] Nan Zhang, Mark Ryan, and Dimitar P. Guelev. 2005. Evaluating Access Control Policies Through Model Checking. In *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3650)*, Jianying Zhou, Javier López, Robert H. Deng, and Feng Bao (Eds.). Springer, 446–460. https://doi.org/10.1007/11556992_32