The Journal of Systems & Software 196 (2023) 111556

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Interaction detection in configurable systems – A formal approach featuring roles☆,☆☆

Philipp Chrszon *, Christel Baier, Clemens Dubslaff, Sascha Klüppelholz

Institute of Theoretical Computer Science, Technische Universität Dresden, Dresden, Germany

ARTICLE INFO

Article history:
Received 19 February 2022
Received in revised form 20 October 2022
Accepted 8 November 2022
Available online 17 November 2022

Keywords:
Formal methods
Model checking
Feature-oriented systems
Software product lines
Role-oriented systems

ABSTRACT

Modern software systems are increasingly complex due to their configurability and adaptivity. For modeling and implementing such systems, the concept of roles is particularly well-suited as it allows capturing context-dependent properties and behavior. Similar to other compositional approaches, notably the feature-oriented development approach, the detection of (unintended) interactions between roles is a challenging task. We consider two new aspects of interactions. *Hierarchical interactions* may occur between systems and *active interplays* describe the actual situations in a sequence of events where components interact. To reason about such interactions, we introduce a compositional modeling framework based on concepts and notions of roles, comprising *role-based automata (RBAs)*. Based on this formal foundation, we present a modeling language for succinctly describing RBAs and an implementation for translating this language into the input language of the probabilistic model checker PRISM. This enables a formal analysis of functional and non-functional properties including system dynamics, context changes, and interactions. We carry out three experimental studies as a proof of concept of such analyses: First, a peer-to-peer protocol study illustrates how undesired hierarchical interactions can be discovered automatically. Second, a study on a self-adaptive production cell demonstrates how undesired interactions influence quality-of-service measures such as reliability and throughput. Third, we illustrate how to incorporate feature-oriented system design in our role-oriented framework by means of the elevator community benchmark system.

1. Introduction

Information and communication systems are a cornerstone of modern society. Driven by the potential of future hardware and the growing demands from users and stakeholders, these systems get steadily more intricate, which renders their design, implementation, and maintenance challenging tasks. This intricacy is raised even further by the customization and adaptation of systems to different customer requirements, country-specific legal regulations, alternative environments, and hardware platforms. There is a clear trend in achieving customization of systems by developing families systems or product lines that build on a core system and combinations of alternative components, rather than manually configuring single software products for the targeted application areas (Clements and Northrop, 2001). Beyond those static adaptation needs, there is a rising demand for dynamic systems that are able to adapt to a changing context at runtime. For instance, the behavior of a system may depend on its location, the time of day, the remaining battery charge, or the network connectivity. The context may also arise from the dynamic collaboration with other systems in vicinity. Furthermore, adaptivity is a crucial requirement for systems possessing self-healing or self-optimization capabilities. In such complex settings, it is simply impossible to foresee all the possible interactions that may occur between subsystems and to predict their impact on the correct operation of the system or its performance. Motivated by the additional complexity imposed by variant-rich and context-dependent dynamic systems, several software-engineering approaches have been developed for taming this complexity.

Features (Kang et al., 1990; Apel and Kästner, 2009) are an established concept to capture the commonalities and differences among system variants, i.e., they provide an abstraction of the variability within a system. Generally, a feature encapsulates optional or incremental functionalities (Zave, 2001). A system variant then corresponds to a combination of features. Within

☆ Editor: Raffaela Mirandola.

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

* Corresponding author.
E-mail addresses: philipp.chrszon@tu-dresden.de (P. Chrszon), christel.baier@tu-dresden.de (C. Baier), clemens.dubslaff@tu-dresden.de (C. Dubslaff), sascha.klueppelholz@tu-dresden.de (S. Klüppelholz).

https://doi.org/10.1016/j.jss.2022.111556
0164-1212/

the feature-oriented development of *software product lines (SPLs)*, the feature concept is applied to promote a systematic reuse of system components called *feature modules* (Apel et al., 2013a). In order to create such system variants defined by a feature combination, the corresponding feature modules are combined with a base system, typically by means of a fully automatic generator tool. In *dynamic feature-oriented systems* (Gomaa and Hussein, 2003) 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 (e.g. Acher et al., 2009; Dubslaff et al., 2015).

Another concept facilitating dynamic runtime adaptations are *roles*. The role concept has first been introduced by Bachman and Daya (1977) in the area of data modeling to capture context-dependent information and the evolution of entities. The main idea is to separate the extrinsic, i.e., context-dependent, properties of an entity from its intrinsic properties and to encapsulate them within roles similar to features encapsulating functionalities in feature-oriented systems. In this regard, the concept of roles can be seen as an extension of features. This is not only conceptually appealing as it enables a more precise modeling of real-world concepts, but it also allows for dealing with transient properties elegantly. For instance, the intrinsic properties "name" and "birthdate" are part of every person's record. A person's office phone number, on the other hand, is dependent on the employment contract and may cease to be valid. Furthermore, multiple part-time employments can be handled easily within a role-oriented approach by creating multiple employee roles for the person. Roles have also been adopted in conceptual modeling where they complement the concepts of objects and relationships (Steimann, 2000; Kühn et al., 2014). Here, roles capture context-dependent properties and behavior. Usually, a role may not only add new behavior to its player, but adapt existing behavior as well. The role concept further facilitates an explicit notion of contexts in which objects act and can also make the relationships between object explicit. Since roles enable fine-grained adaptations on the level of individual components or objects, they are particularly suited for the modeling and implementation of context-sensitive adaptive systems.

In this paper, we present a formal approach to model and analyze role-oriented systems. Our view on role-oriented systems takes inspiration from the *Compartment Role Object Model (CROM)* (Kühn et al., 2015) that unifies most of the well-established views on roles from the literature and provides a graphical notation similar to UML class diagrams (OMG, Object Management Group, 2011). This meta-model provides the concepts of *naturals*, *roles*, and *compartments*, where naturals constitute the players of roles and compartments are objectified contexts in which roles are played. Consider the example shown in Fig. 1(a), where different stations (the naturals) play the roles of clients and servers in different networks (the compartments). Using a purely feature-oriented approach, the system of networks could be represented using the feature model shown in Fig. 1(b) as a feature diagram. Here, the roles of a station correspond to (dynamic) features, which can be activated depending on whether the station acts as a client or a server. Note that a natural can also play multiple roles (even possibly of the same type) in different compartments, like $\mathsf{Station}_2$ in the example, which allows us to easily model interactions between different systems represented by compartments. While features are well known for their capability of modeling contexts (Mauro et al., 2016; Dubslaff et al., 2019), they cannot directly model this simultaneous role-playing. For describing this behavior in the feature-oriented modeling approach, the features corresponding to the shared naturals would have to be replicated for each compartment and their behavior synchronized, turning conceptual modeling to rely on implementation details. Utilizing the concept of multi product lines (Holl et al., 2012; Trujillo-Tzanahua et al., 2018) to represent each compartment as an individual product line only partially alleviates the issue, since the merging of the shared feature corresponding to the naturals still needs special treatment (Schröter et al., 2016). Furthermore, multi product lines are mainly tailored towards a static composition of product lines, while the role concept is particularly well-suited for dynamic collaborations of (sub-)systems.

A major challenge during the development of systems incorporating features are *feature interactions* (Plath and Ryan, 2001; Calder et al., 2003; Apel et al., 2013b). Such interactions describe behavior that arises from the combination of multiple features and are not immediately deducible from their individual behaviors. They are often desired and unavoidable for defining the behavior of the system via composition and enable a modularization of feature-specific behavior. However, interactions may also be unintended, leading to unforeseen side effects that may compromise the functional correctness of the system or negatively affect its non-functional properties, such as energy consumption, throughput, or reliability. There is a substantial corpus of work on detecting feature interactions (e.g. Kuhn et al., 2004; Yilmaz et al., 2006; Qu et al., 2008; Garvin and Cohen, 2011; Dubslaff et al., 2022), witnessing their importance in feature-oriented system design. This importance can also be expected in systems designed by a role-oriented approach, since these systems can be seen as an extension of feature-oriented systems.

One of the main contributions of this paper is the use of our developed formal framework for role-oriented systems to detect and analyze interactions between roles. In particular, we introduce two new aspects regarding interactions, namely *hierarchical interactionss* and *active interplayss*. Traditionally, interactions are only considered within individual systems. However, complex systems are often organized hierarchically, thus there may not only be interactions within a system, but also interactions between systems. We call this new kind of interactions hierarchical interactionss. In the example in Fig. 1, an interaction between the client and server roles played by $\mathsf{Station}_2$ within different compartments would constitute such an inter-system interaction. Active interplays address the challenge of pinpointing the actual situations where interactions happen during execution time of systems. Especially in highly dynamic adaptive systems, these time points when an interaction takes place and which roles were played to cause that interaction are of interest. To reason about such active interplayss, it is necessary that the active participation of a role in an event or action is detectable, e.g., by means of role-annotated implementations or operational system models. We approach a formal analysis of interactions that also incorporate quantitative aspects, e.g., interactions that lead to the increase of energy consumption or probability of failure, by the use of *probabilistic model checking* (Kwiatkowska et al., 2002; Baier and Katoen, 2008).

Approach and contributions. We propose a formal modeling and analysis approach for adaptive systems using the concept of roles as first-class construct. Specifically, we present

(1) *role-based automata* (RBAs) for the representation and coordination of role-specific operational behavior with operators for role-binding and their parallel execution,
(2) a modeling language to specify the operational behavior of systems with role constructs,
(3) an automated translation of our modeling language to the input language of the probabilistic model checker Prism (Kwiatkowska et al., 2002), and

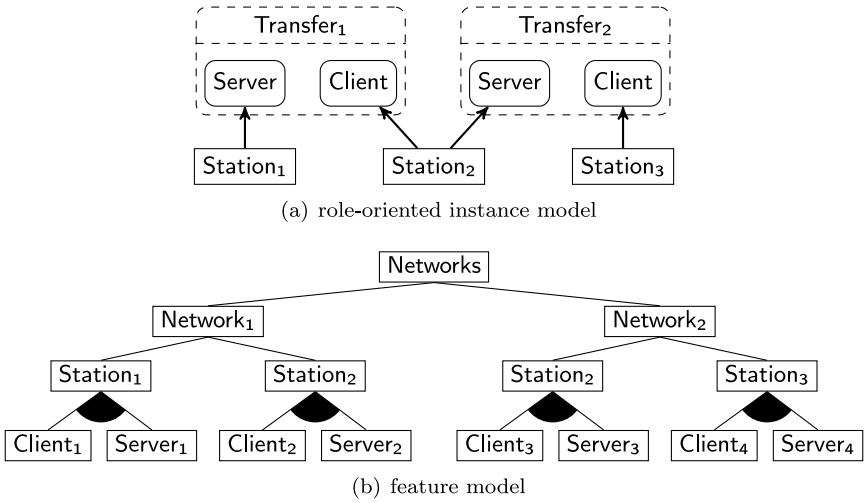


Fig. 1. A set of stations playing the roles of client and server in different networks and a possible feature-oriented model for the system of networks.

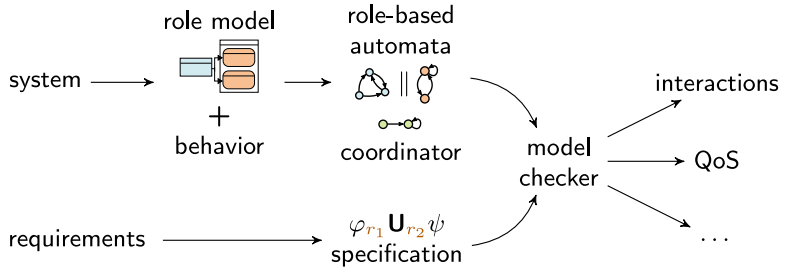


Fig. 2. Overview of the analysis approach.

(4) experimental studies to illustrate how our translation can be used to analyze RBAs for detecting hierarchical interactionss and active interplayss, their impact on functional correctness, and quantitative measures.

Ad (1): We define a compositional framework to model role-oriented systems by introducing *role-based automata (RBAs)*. RBAs provide a uniform formalism to model the systems components by means of *naturals*, *roles*, and *compartments*. On RBAs we define composition operators that realize the effect of role-binding on the structural level. Our framework is inspired by feature-oriented specification formalisms such as *featured transition systems (FTSs)* (Classen et al., 2013) and its probabilistic variants (Dubslaff et al., 2015; Chrszon et al., 2018; Vandin et al., 2018). To this end, our composition operators also follow the concepts established in the area of feature-oriented system design, namely superimposition (Chandy and Misra, 1988; Plath and Ryan, 2001; Dubslaff, 2019) and parallel feature composition (Classen et al., 2013; Dubslaff et al., 2015; Chrszon et al., 2018). An RBA is a state-based model where transitions are guarded with role annotations that describe 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 stochastic choice is present, for instance, in cryptographic algorithms as well as in coordination and communication protocols, external stochastic effects 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)* (Puterman, 1994) that encodes role annotations into its actions and thus can be used towards a quantitative analysis of role-specific and context-dependent behavior.

Ad (2): To specify RBAs and coordinators, we introduce a lightweight modeling language based on guarded commands (Jifeng et al., 1997) and reactive modules (Alur and Henzinger, 1999). Our language is in the line of well-established guarded command languages used to describe the operational behavior of feature-oriented systems (Classen et al., 2013; Dubslaff et al., 2015) such as FPROMELA (Classen et al., 2012) and PROFEAT (Chrszon et al., 2018). Hence, by the aforementioned correspondences, our language can be used to specify both systems incorporating features and roles.

Ad (3): To exemplify how the RBA formalism can be utilized to analyze possibly hierarchic system models, we implemented an automated translation of our role-oriented modeling language into the input language of the probabilistic model checker PRISM (Kwiatkowska et al., 2002). The PRISM language is supported by a wide range of quantitative analysis tools (e.g. Kwiatkowska et al., 2002; Dehnert et al., 2017). Thus, our transformation unleashes the full power of functional and quantitative analysis methods also for systems defined in our modeling language (2). In Fig. 2, the general schema of the analysis approach is depicted.

Ad (4): We demonstrate and evaluate our analysis approach and tooling by three experimental studies. First, we consider a peer-to-peer file-transfer protocol. Our formal analysis reveals undesired (functional) interactions between different networks, illustrating role interaction and hierarchical interactions detection. Furthermore, our tooling reports on role-annotated action traces as witness for the violation of certain properties. This enables to pinpoint the situations where interactions actually take place (cf. role-playing and active interplays detection). Second, we consider a production-cell system where we focus on quantitative hierarchical interactionss by evaluating reliability and throughput. Within this experimental study, we also demonstrate how self-adaptive systems can be modeled analyzed within our

framework using RBAs and role-playing coordinators. Third, we illustrate how the well-known elevator system from the feature-oriented systems community (Plath and Ryan, 2001; Chrszon et al., 2018) is modeled and analyzed with our role-oriented formalism, demonstrating the benefits of roles compared to features and the use of our role-interaction detection mechanisms for isolating feature interactions.

This paper is an extended version of our previous publication (Chrszon et al., 2020). Besides providing proofs which were not included in the conference version, we present the framework in the quantitative setting by means of probabilistic RBAs. Additionally, we extended the material mainly in the following regards:

- We provide an in-depth description of the modeling language and elaborate on design decisions.
- We detail implementation challenges and solutions.
- We formally establish embeddings into and from featured MDPs (Dubslaff et al., 2015) from and into a subclass of RBAs, showing that RBAs in fact extend featured MDPs and thus also FTSs.
- An additional experimental study modeling an elevator system provides more insights on the connection between feature- and role-oriented systems.

The implementation and the experiments contained in this article are provided in the accompanied artifact (Anon, 2022).

2. Preliminaries

Let us first introduce notations and formal foundations used throughout the paper. If not stated differently, sets are assumed to be finite. For a set X, we denote by 2^X the power set of X. Given a function $f : X \to Y$ and a set $Z \subset X$, we denote by $f(Z)$ the element-wise union of f, i.e., $f(Z) = \{f(z) : z \in Z\}$.

A probability *distribution* over a set X is a function $\delta : X \to [0, 1]$ with $\sum_{x \in X} \delta(x) = 1$. We denote the set of distributions over X by $Distr(X)$. The *support* of a distribution δ is defined as the set $\{x \in X : \delta(x) > 0\}$. A *Dirac distribution* is a distribution δ for which there is an $x \in X$ with $\delta(x) = 1$. We denote by $Dirac_X(x) \in Distr(X)$ the uniquely defined Dirac distribution where $Dirac_X(x)(x) = 1$ for $x \in X$. When clear from the context, we might drop the subscript X and write $Dirac(x)$ instead of $Dirac_X(x)$.

Let X and Y be two sets and $\delta_X \in Distr(X)$ and $\delta_Y \in Distr(Y)$ be two distributions over X and Y respectively. The *product distribution* of δ_X and δ_Y, denoted $\delta_X * \delta_Y \in Distr(X \times Y)$, is defined as $(\delta_X * \delta_Y)(x, y) = \delta_X(x) \cdot \delta_Y(y)$ for all $x \in X$, $y \in Y$.

Definition 1 (*MDP*)**.** A *Markov decision process* is a tuple $\mathcal{M} = (S, Act, \longrightarrow, S^{init})$ where S is a finite set of states, Act is a finite set of actions, $\longrightarrow \subseteq S \times Act \times Distr(S)$ is a transition relation, and $S^{init} \subseteq S$ is a set of initial states.

We use the notation $s \xrightarrow{\alpha} \lambda$ for $(s, \alpha, \lambda) \in \longrightarrow$. In case for all $s \xrightarrow{\alpha} \lambda$ we have that λ is a Dirac distribution, we call the MDP *transition system*.

Multi-action MDPs are an extension of MDPs where the transition relation includes sets of actions rather than single actions (Baier et al., 2018).

Definition 2 (*maMDP Baier et al., 2018*)**.** A *multi-action MDP* is a tuple $\mathcal{M} = (S, Act, \longrightarrow, S^{init})$ where S is a finite set of states, Act is a set of actions, $\longrightarrow \subseteq S \times 2^{Act} \times Distr(S)$ is a transition relation with multi-actions, and $S^{init} \subseteq S$ is a set of initial states.

Definition 3 ($\cong$ *for maMDPs*)**.** Two maMDPs $\mathcal{M}_i = (S_i, Act_i, \longrightarrow_i, S_i^{init})$ for $i \in \{1, 2\}$ are *equivalent up to isomorphism*, denoted $\mathcal{M}_1 \cong \mathcal{M}_2$, if $Act_1 = Act_2$ and there is a bijection $\xi : S_1 \to S_2$ such that

- $\xi(S_1^{init}) = S_2^{init}$, and
- $s \xrightarrow{\Sigma}_1 \lambda$ iff $\xi(s) \xrightarrow{\Sigma}_2 \xi(\lambda)$ for all $s \in S_1$, $\Sigma \subseteq Act_1$, and $\lambda \in Distr(S_1)$

where $\xi(\lambda) \in Distr(S_2)$ is defined as $\xi(\lambda)\big(\xi(s)\big) = \lambda(s)$ for all $s \in S_1$.

The introduction of multi-actions requires an adaptation of the standard parallel-composition operator for MDPs. The parallel composition defined in the following is derived from the composition of data-abstract *simple probabilistic constraint automata* (spCA) (Baier, 2005).

Definition 4 (*Parallel maMDP Composition*)**.** The *parallel composition* of two multi-action MDPs $\mathcal{M}_i = (S_i, Act_i, \longrightarrow_i, S_i^{init})$ with $i \in \{1, 2\}$ for a set of non-synchronizing actions $N \subseteq Act_1 \cup Act_2$ is defined as

$$\mathcal{M}_1 \parallel_N \mathcal{M}_2 = \left(S_1 \times S_2,\ Act_1 \cup Act_2,\ \longrightarrow,\ S_1^{init} \times S_2^{init}\right)$$

where $\longrightarrow$ is the smallest transition relation fulfilling the rules shown in Fig. 3.

Note that the (sync) rule poses no restrictions regarding the emptiness of $\Sigma_1 \cap Act_2$ or $\Sigma_2 \cap Act_1$. Thus, the maMDPs may synchronize over non-shared actions. A slight extension compared to the composition of spCA is the inclusion of non-synchronizing actions.

3. Modeling role-oriented behavior

We introduce *role-based automata* (RBAs) as uniform formalism to model the essential building blocks of role-oriented systems: naturals, roles, and compartments. Let us illustrate the intuition behind these building blocks by an everyday example from the football domain. *Naturals* provide the atomic elements indivisible on the chosen level of abstraction, e.g., football players. They can admit *roles* and act accordingly, e.g., positions such as midfielder or defender a football player can play. *Compartments* provide a mechanism to model contexts in which roles are played, e.g., the teams the football player is playing in. They can coordinate the role-playing, e.g., the player plays midfielder in the home team but defender in the national team. We also allow for role-playing of compartments, e.g., modeling the role of the whole football team in a competition. To this end, hierarchical structures of role playing can be modeled (Chrszon et al., 2016): A player plays the midfielder in the national team, which plays in group B of the world cup, which itself plays a role in the world-wide sport events of the year.

RBAs modeling these building blocks can be combined by *role-binding* and *parallel-composition* operators. Intuitively, the role-binding operator includes role-specific behaviors to RBA descriptions of their players, i.e., naturals and compartments, and parallel composition is used to combine the behaviors of all players with their possible roles towards a description of the overall behavior of the role-oriented system. Parallel composition of RBAs formalizes the communication between RBAs using the standard notion of synchronization over shared actions. A key concept in role-oriented 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 (Mizoguchi et al., 2012). However, there are usually constraints on the role-playing, formalized using another automata-based component, the *(role-playing) coordinator*. The composition of an RBA

$$(\text{int}_1)\frac{s_1 \xrightarrow{\Sigma}_1 \lambda_1 \qquad \Sigma \cap Act_2 = \varnothing}{\langle s_1, s_2\rangle \xrightarrow{\Sigma} \lambda_1 * Dirac(s_2)} \qquad (\text{int}_2)\frac{s_2 \xrightarrow{\Sigma}_2 \lambda_2 \qquad \Sigma \cap Act_1 = \varnothing}{\langle s_1, s_2\rangle \xrightarrow{\Sigma} Dirac(s_1) * \lambda_2}$$

$$(\text{sync})\frac{s_1 \xrightarrow{\Sigma_1}_1 \lambda_1 \qquad s_2 \xrightarrow{\Sigma_2}_2 \lambda_2 \qquad \Sigma_1 \cap Act_2 = \Sigma_2 \cap Act_1 \qquad |(\Sigma_1 \cup \Sigma_2) \cap N| \leq 1}{\langle s_1, s_2\rangle \xrightarrow{\Sigma_1 \cup \Sigma_2} \lambda_1 * \lambda_2}$$

Fig. 3. Rules for the parallel composition of multi-action MDPs.

with a coordinator resolves all the allowed combinations of role-playing and yields a Markov decision process (MDP). Our semantics also captures the case without probabilistic choices (Chrszon et al., 2020), where the semantics coincides with a transition system. As transition systems and MDPs are well-established formalisms with broad applications, this enables the use of analysis tools to reason about systems involving roles.

Running example. In the following, we formally provide definitions of RBAs and our operators on RBAs. Along with the formal definitions, we illustrate role-oriented concepts by means of a simple banking example adapted from Coplien and Reenskaug (2014). 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 modeled as RBAs and 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, also modeled as RBAs.

3.1. Role-based automata

Before we formally provide the definition of RBAs, we need technical notions of role interfaces and annotations.

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 $U = \varnothing$, 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 roles, i.e., $(B_1 \cup U_1) \cap (B_2 \cup U_2) = \varnothing$, 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 $\mathbb{A}(R) = \{r, \bar{r}, +r : r \in R\}$, where transitions annotated with r or $\bar{r}$ stand for role r is actively played, or explicitly not played, respectively. In case a transition is neither annotated with r nor $\bar{r}$, 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 5 (*RBA*)**.** A *role-based automaton* is a tuple

$$\mathcal{A} = \left(S, Act, R, \longrightarrow, S^{init}\right)$$

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

Note that this definition formalizes RBAs as a probabilistic automaton (Chrszon et al., 2020). In a transition (s, α, X, λ) 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 λ is a distribution over S representing an internal probabilistic choice of the successor state. We assume that for all transitions and roles $r \in R$ we have that $|X \cap \{r, \bar{r}, +r\}| \leq 1$ and write $s \xrightarrow{\alpha/X} \lambda$ for $(s, \alpha, X, \lambda) \in \longrightarrow$. An RBA is an *unbound role instance* of a role r if $R = \langle \varnothing, \{r\}\rangle$ and all transitions are annotated with r, $\bar{r}$, or $+r$. If the role interface of an RBA is empty, i.e., $R = \varnothing$, it represents a natural.

Running example. Towards our running example of a banking account with money transfer, we define four RBAs, modeling the two accounts Acc_i for $i \in \{1, 2\}$ between which money is transferred and source and target roles for these accounts, denoted *Source* and *Target*, respectively. Fig. 4a shows the RBA for the accounts Acc_i, $i \in \{1, 2\}$. To keep the example automata small, the account balance is either 0 or 1 and can be increased or decreased using the *inc* and *dec* actions, respectively. Since this RBA represents a natural, all role annotations are empty. In Fig. 4b, the RBA representing the source account role is shown. In the ready state (R_s), the withdraw action w may be invoked, which enters the processing state (P_s). Self-loops annotated with $\{\bar{s}\}$ enable inc_1 and dec_1 actions if the *Source* role s is not played. Note that the RBA in Fig. 4b is an unbound role instance of s. Fig. 4c shows the *Target* role automaton, which essentially is obtained from the *Source* role automaton by replacing role annotations s with t, the reference account 1 with 2, *dec* actions with *inc* actions, and withdraw w with deposit d.

3.1.1. Parallel composition

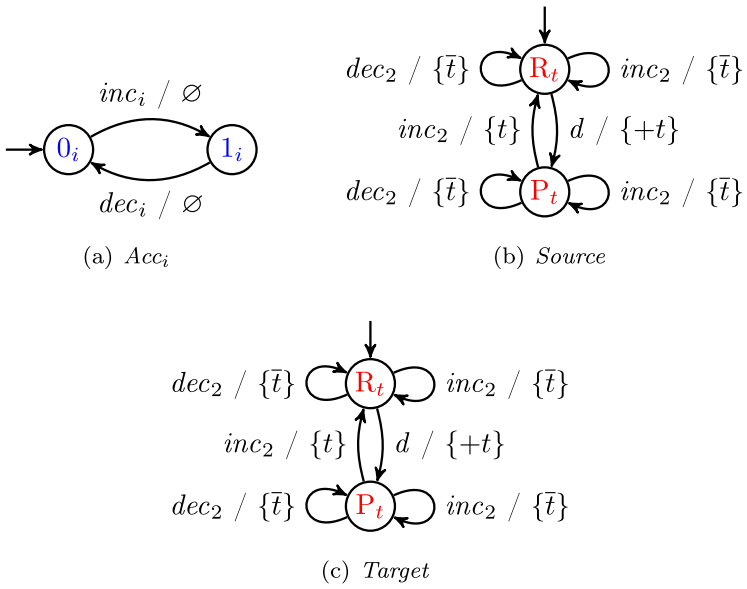
The *parallel-composition* operator on RBAs extends the well-known definition of parallel composition on MDPs (cf. Definition 4) with role annotations. This is done in a similar way as parallel composition on featured transition systems (FTSs, Classen et al. (2013)) and its probabilistic variants (Dubslaff et al., 2015; Chrszon et al., 2018), extending parallel composition of transition systems and MDPs with feature annotations. Two automata are running concurrently and communicate via handshaking, i.e., synchronization over shared actions.

Definition 6 (*Parallel Composition*)**.** 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

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

where $\longrightarrow$ is the smallest transition relation fulfilling the rules shown in Fig. 5.

Composing two RBAs again yields an RBA. In the case of interleaving (rules int_1 and int_2), the *Dirac* distribution ensures that one of the RBAs remains in its original state while the other one takes its transition. Note that through joining the role annotations in the (sync) rule, multiple roles could be played in parallel when taking a single transition.



(a) Acc_i (b) *Source* (c) *Target*

Fig. 4. Role-based automata for the account naturals and source and target role with *inc*rement, *dec*rement, *d*eposit, and *w*ithdraw actions.

$$(\text{int}_1)\ \frac{s_1 \xrightarrow{\alpha/X}_1 \lambda_1 \qquad \alpha \in Act_1 \setminus Act_2}{\langle s_1, s_2\rangle \xrightarrow{\alpha/X} \lambda_1 * Dirac(s_2)} \qquad (\text{int}_2)\ \frac{s_2 \xrightarrow{\alpha/Y}_2 \lambda_2 \qquad \alpha \in Act_2 \setminus Act_1}{\langle s_1, s_2\rangle \xrightarrow{\alpha/Y} Dirac(s_1) * \lambda_2}$$

$$(\text{sync})\ \frac{s_1 \xrightarrow{\alpha/X}_1 \lambda_1 \qquad s_2 \xrightarrow{\alpha/Y}_2 \lambda_2 \qquad \alpha \in Act_1 \cap Act_2}{\langle s_1, s_2\rangle \xrightarrow{\alpha/X\cup Y} \lambda_1 * \lambda_2}$$

Fig. 5. Rules for the parallel composition of RBAs.

As mentioned before, compartments mainly serve as a structural element and logically represent the context in which the contained roles are played. Within the RBA framework, a compartment is formed by the parallel composition of its unbound role instances.

3.1.2. Role binding

The *role-binding* operator combines the behavior of a role with its player, enabling the player to play the bound role.

Definition 7 (*Role Binding*)**.** 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 $R = R_a \oplus R_p \oplus \langle\{r\}, \varnothing\rangle$ and where $\longrightarrow$ is the smallest transition relation fulfilling the rules shown in Fig. 6.

Note that the first three rules in Fig. 6 are defined as for the parallel composition (see Definition 6). Rule (add) covers the case where the role 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 role r to add new behavior to its player using the role annotation $+r$, similar to the concept of superimposition in Dubslaff (2019).

In our running example, the *Target* role in Fig. 4c adds the action d to its player. The result of binding the *Target* role to the player Acc_2 (see Fig. 4) is shown in Fig. 7. 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 Fig. 4c 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 $Source[s \to Target[t \to Acc_2]]$.

3.1.3. Algebraic properties of compositions

The composition operators defined above fulfill certain useful algebraic properties. We first define an equivalence relation on RBAs.

Definition 8 ($\cong$ *for RBAs*)**.** Two RBAs $\mathcal{A}_i = (S_i, Act_i, R_i, \longrightarrow_i, S_i^{init})$ for $i \in \{1, 2\}$ are *equivalent up to isomorphism*, denoted $\mathcal{A}_1 \cong \mathcal{A}_2$, if $Act_1 = Act_2$, $R_1 = R_2$, and there is a bijection $\xi : S_1 \to S_2$ such that

- $\xi(S_1^{init}) = S_2^{init}$, and
- $s \xrightarrow{\alpha/X}_1 \lambda$ iff $\xi(s) \xrightarrow{\alpha/X}_2 \xi(\lambda)$ for all $s \in S_1$, $\alpha \in Act_1$, $X \in \mathbb{A}(R_1)$, and $\lambda \in Distr(S_1)$

where $\xi(\lambda) \in Distr(S_2)$ is defined as $\xi(\lambda)\big(\xi(s)\big) = \lambda(s)$ for all $s \in S_1$.

Based on this definition of equivalence, we can establish the following properties on RBAs:

Theorem 1. *For pairwise compatible RBAs $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{P}_1$, and $\mathcal{P}_2$ where $\mathcal{A}_1$ and $\mathcal{A}_2$ are unbound role instances with names a_1 and a_2,*

$$(\text{int}_a)\ \frac{s_a \xrightarrow{\alpha/X}_a \lambda_a \qquad \alpha \in Act_a \backslash Act_p}{\langle s_a, s_p\rangle \xrightarrow{\alpha/X} \lambda_a * Dirac(s_p)} \qquad (\text{int}_p)\ \frac{s_p \xrightarrow{\alpha/Y}_p \lambda_p \qquad \alpha \in Act_p \backslash Act_a}{\langle s_a, s_p\rangle \xrightarrow{\alpha/Y} Dirac(s_a) * \lambda_p}$$

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

$$(\text{add})\ \frac{s_a \xrightarrow{\alpha/X}_a \lambda_a \qquad \alpha \in Act_a \cap Act_p \qquad +r \in X}{\langle s_a, s_p\rangle \xrightarrow{\alpha/X\backslash\{+r\}\cup\{r\}} \lambda_a * Dirac(s_p)}$$

Fig. 6. Rules for binding a role r within an RBA $\mathcal{A}$ to a player $\mathcal{P}$.

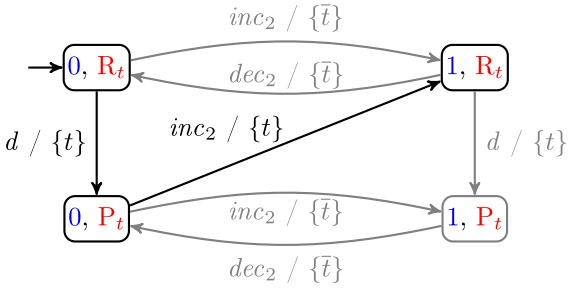


Fig. 7. Result of binding *Target* to Acc_2 ($Target[t \to Acc_2]$).

respectively, we have

$$\mathcal{A}_1 \parallel \mathcal{A}_2 \cong \mathcal{A}_2 \parallel \mathcal{A}_1 \tag{1}$$

$$\mathcal{A}_1 \parallel (\mathcal{A}_2 \parallel \mathcal{A}_3) \cong (\mathcal{A}_1 \parallel \mathcal{A}_2) \parallel \mathcal{A}_3 \tag{2}$$

$$(\mathcal{A}_1 \parallel \mathcal{A}_2)[a_1 \to \mathcal{P}_1] \cong \mathcal{A}_1[a_1 \to \mathcal{P}_1] \parallel \mathcal{A}_2 \tag{3}$$

$$(\mathcal{A}_1 \parallel \mathcal{A}_2)[a_1 \to \mathcal{P}_1][a_2 \to \mathcal{P}_2] \cong (\mathcal{A}_1 \parallel \mathcal{A}_2)[a_2 \to \mathcal{P}_2][a_1 \to \mathcal{P}_1] \tag{4}$$

In case further $Act_1 \cap Act_2 = \varnothing$ for the action sets Act_i of $\mathcal{A}_i$, $i \in \{1, 2\}$, we have

$$\mathcal{A}_1[a_1 \to \mathcal{A}_2[a_2 \to \mathcal{P}_1]] \cong \mathcal{A}_2[a_2 \to \mathcal{A}_1[a_1 \to \mathcal{P}_1]] \tag{5}$$

Proof. Let $\mathcal{A}_i = (S_i, Act_i, R_i, \longrightarrow_i, S_i^{init})$ and $\mathcal{P}_j = (S_{\mathcal{P}_j}, Act_{\mathcal{P}_j}, R_{\mathcal{P}_j}, \longrightarrow_{\mathcal{P}_j}, S_{\mathcal{P}_j}^{init})$ be RBAs for $i \in \{1, 2, 3\}$ and $j \in \{1, 2\}$.

Ad (1): Commutativity of the parallel composition is clear since $\cup$ and $*$ (the product measure on distributions) are both commutative. Furthermore, the conditions in the premise of (sync) are symmetric and (int_1) is symmetric to (int_2). Compatibility of the role interfaces is also clear by the commutativity of interface compatibility.

Ad (2): Associativity of the parallel composition of RBAs follows directly from the associativity of the parallel composition for MDPs. The only difference are the additional role-playing annotations. However, associativity is also given here by the associativity of $\cup$. It remains to show compatibility of the role interfaces. From the left hand side, we obtain $R_2 \cap R_3 = \varnothing$ and $R_1 \cap (R_2 \cup R_3) = \varnothing$. Thus, $R_1 \cap R_2 = \varnothing$ and $R_1 \cap R_3 = \varnothing$. Therefore, $(R_1 \cup R_2) \cap R_3 = \varnothing$.

Ad (3): Note that the rules for the role-binding operator ($\to$) are equivalent to the rules for parallel composition in case that a transition is not labeled with a $+r$ annotation. Thus, for proving the property in that case, we can rely on the associativity of parallel composition (2). It remains to show that (3) holds in case $\mathcal{A}_1$ has transitions labeled with $+a_1$. Assume there is a transition $s_1 \xrightarrow{\alpha/X\cup\{+a_1\}}_1 \lambda_1$. We have to consider the cases where $\alpha \in Act_1 \backslash Act_2$ (int_1) and $\alpha \in Act_1 \cap Act_2$ (sync):

(int$_1$): Applying rule (add) on the left hand side yields for all $s_2 \in S_2$ and $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ transitions

$$\langle s_1, s_2, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_2) * Dirac(s_{\mathcal{P}_1}) .$$

On the right hand side, binding a_1 in $\mathcal{A}_1$ to $\mathcal{P}_1$ yields for all $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ the transitions

$$\langle s_1, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_{\mathcal{P}_1}) .$$

Then, the parallel composition with $\mathcal{A}_2$ yields

$$\langle s_1, s_{\mathcal{P}_1}, s_2\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_{\mathcal{P}_1}) * Dirac(s_2) .$$

(sync): Applying rule (add) on the left hand side yields for all $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ transitions

$$\langle s_1, s_2, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * \lambda_2 * Dirac(s_{\mathcal{P}_1}) .$$

On the right hand side, binding a_1 in $\mathcal{A}_1$ to $\mathcal{P}_1$ yields for all $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ the transitions

$$\langle s_1, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_{\mathcal{P}_1}) .$$

Then, the parallel composition with $\mathcal{A}_2$ yields

$$\langle s_1, s_{\mathcal{P}_1}, s_2\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_{\mathcal{P}_1}) * \lambda_2 .$$

Ad (4): For proving this property, we rely on the commutativity of parallel composition (1). Remind that the rules for role binding correspond to the rules for parallel composition in case a transition is not labeled with a $+r$ annotation. Commutativity of the (add) rule is clear since $(X \setminus \{+r_1\} \cup \{r_1\}) \setminus \{+r_2\} \cup \{r_2\} = (X \setminus \{+r_2\} \cup \{r_2\}) \setminus \{+r_1\} \cup \{r_1\}$.

Ad (5): For the rules (int_a), (int_p), and (sync) property (5) is clear by associativity (2) and commutativity (1) of the parallel composition. It remains to show that (5) holds if $\mathcal{A}_1$ or $\mathcal{A}_2$ have overriding transitions. Assume $\mathcal{A}_1$ has a transition $s_1 \xrightarrow{\alpha/X\cup\{+a_1\}} \lambda_1$ with $\alpha \in Act_1 \cap Act_{\mathcal{P}_1}$. Then, binding $\mathcal{A}_1$ on the left hand side yields for all $s_2 \in S_2$ and $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ transitions

$$\langle s_1, s_2, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_2) * Dirac(s_{\mathcal{P}_1}) .$$

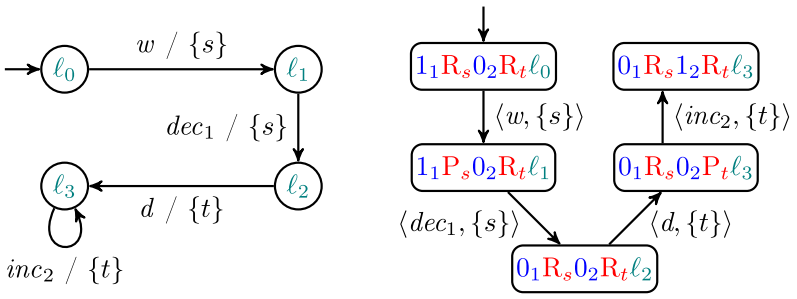
On the right hand side, binding $\mathcal{A}_1$ yields for all $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ transitions

$$\langle s_1, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} \lambda_1 * Dirac(s_{\mathcal{P}_1}) .$$

Since $Act_1 \cap Act_2 = \varnothing$, there is no synchronization of $\mathcal{A}_2$'s transitions with the above transitions introduced by binding $\mathcal{A}_1$. Thus, binding $\mathcal{A}_2$ yields for all $s_2 \in S_2$ and $s_{\mathcal{P}_1} \in S_{\mathcal{P}_1}$ transitions

$$\langle s_2, s_1, s_{\mathcal{P}_1}\rangle \xrightarrow{\alpha/X\backslash\{+a_1\}\cup\{a_1\}} Dirac(s_2) * \lambda_1 * Dirac(s_{\mathcal{P}_1}) .$$

The case where $\mathcal{A}_2$ has an overriding transition can be shown as for $\mathcal{A}_1$ by swapping $\mathcal{A}_1$ and $\mathcal{A}_2$. □



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

Fig. 8. Role-playing coordinator and transition system for $[\![(Source[s \to Acc_1] \parallel Target[t \to Acc_2])]\!]_{\mathsf{C}}$.

3.1.4. Non-blocking roles

As apparent in the (sync) rule of Fig. 6, 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 composition of feature modules of feature-oriented systems (Chrszon et al., 2018; Dubslaff, 2019). We call a role r *non-blocking* for action α if every state s of the associated RBA contains a self-loop $(s, \alpha, \{\bar{r}\}, Dirac(s))$. The role *Target* in Fig. 4c provides such a self-loop for each action of player Acc_2. Thus, the original behavior of Acc_2 is preserved upon role binding (cf. horizontal transitions in Fig. 7).

3.2. 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. A role annotation in an RBA may stand for multiple role-playings. To introduce an explicit notion of role-playings, we define the 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 $\bar{r} \notin X$, and
(2) for all $r \in R$ with $\{r, +r\} \cap X \neq \varnothing$ 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.

3.2.1. Role-playing coordinator

We specify the semantics of RBAs w.r.t. a coordination component called *(role-playing) coordinator* that provides the rules for role-playing in an RBA. Essentially, a coordinator is an instance of an RBA where role annotations are interpreted as restrictions to role-playing.

Definition 9 (*Coordinator*)**.** A *role-playing coordinator* is an RBA $\mathsf{C} = \left(S, Act, R, \longrightarrow, S^{init}\right)$ where R is closed and for all $\ell \xrightarrow{\langle \alpha, Y\rangle} \lambda$ and $r \in R$ we have $+r \notin Y$.

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 Fig. 8a 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 $\{s, t\}$.

3.2.2. Semantics of RBAs

Given an RBA $\mathcal{A}$ and a coordinator C, the operational semantics of $\mathcal{A}$ under C is a Markov decision process (MDP).

Definition 10 (*MDP Semantics w.r.t. Coordinator*)**.** 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 an MDP

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

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 following rules shown in Fig. 9.

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, an MDP semantics for RBAs is provided by resolving every possible role-playing nondeterministically due to (int_a) in being the only rule applicable. Fig. 8b shows the MDP for the banking example with two accounts, one playing the *Source* role and the other playing the *Target* role, under the coordinator C shown in Fig. 8a. The coordinator realizes that money is withdrawn from the source account by decrementing its balance and then depositing money on the target account.

Having MDPs as underlying semantics for role-oriented systems allows us to apply methods for simulation and verification, such as standard model-checking algorithms for temporal logics such as PCTL and LTL (Baier and Katoen, 2008) or statistical model-checking (Legay et al., 2010). To reason about role-playing, which is encoded into the actions of the resulting MDP, standard approaches using action-based logics may be employed (De Nicola and Vaandrager, 1990).

3.3. Relation to featured MDPs and features transition systems

In essence, RBAs are used for modeling the operational behavior of role-oriented systems, similar to how featured transition systems (FTS) (Classen et al., 2013) and its probabilistic variants (Dubslaff et al., 2015; Chrszon et al., 2018) are used for modeling the behavior of feature-oriented systems. To this end, role-binding is similar to weaving a feature's implementation into a base system (Plath and Ryan, 2001; Dubslaff et al., 2015; Chrszon et al., 2018; Dubslaff, 2019), while parallel composition of RBAs corresponds to the parallel composition of isolated feature-oriented systems (Classen et al., 2013). The main difference between our framework relying on RBAs and its feature-oriented counterparts is the capability to model compartments

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

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

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

Fig. 9. Rules for the semantics of RBAs w.r.t. coordinator RBAs.

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 Chrszon et al. (2016) where coordination principles in role-oriented systems were considered.

In order to illustrate the expressiveness of RBAs compared to probabilistic variants of FTSs by means of featured MDPs (fMDPs), we present an embedding of fMDPs into RBAs. This in particular shows that our RBA formalism extends the fMDP and thus also the FTS formalisms.

Definition 11 (*Featured MDP Dubslaff et al., 2015*)**.** A *featured Markov decision process* is a tuple $\text{Fmdp} = (S, F, Act, \text{P}, I)$ where S is a finite set of states, F is a set of features, Act is a set of actions, $\text{P} \subseteq S \times \mathbb{B}(F) \times Act \times Distr(S)$ is a featured probabilistic transition relation, and $I \subseteq S$ is the set of initial states. Here, $\mathbb{B}(F)$ stands for the set of Boolean expressions over the features in F.

The main idea for the transformation of an fMDP to an RBA is to treat a selected feature as an actively played role. Since a feature guard is Boolean expression over features, it may represent multiple feature combinations and thus potentially corresponds to multiple role annotations. We define the *corresponding role annotation of a feature combination* $p \in 2^F$ over the set of features F as $X(p) = \{f : f \in p\} \cup \{\bar{f} : f \notin p\}$.

Definition 12 (*RBA Induced by An fMDP*)**.** The *corresponding RBA of an fMDP* $\text{Fmdp} = (S, F, Act, \text{P}, I)$ is defined as

$$\mathcal{A}[\text{Fmdp}] \ = \ \big(S,\ Act,\ \langle F, \varnothing\rangle,\ \longrightarrow,\ I\big)$$

where $\longrightarrow$ is the smallest transition relation fulfilling the following rule:

$$\frac{(s,g,\alpha,\lambda) \in \text{P} \qquad p \models g \qquad p \in 2^F}{s \xrightarrow{\alpha/X(p)} \lambda}$$

Note that any FTS can be transformed straightforwardly into an fMDP by turning each transition into probabilistic transition with a Dirac distribution. Hence, the embedding of fMDPs into RBAs also provides an embedding of FTS into RBA. In combination with a feature model such as feature diagrams (Kang et al., 1990), the resulting RBA is a *family model* that represents the behaviors of every system variant. Towards a family-based analysis, we can apply *configuration lifting* (Post and Sinz, 2008) by encoding the set of valid feature combinations into the state space of the coordinator.

Definition 13 (*Coordinator Arising from a Feature Model*)**.** The *coordinator arising from a feature model* $d \subseteq 2^F$ is defined as

$$\text{C} \ = \ \big(d,\ Act,\ \langle F, \varnothing\rangle,\ \longrightarrow,\ d\big)$$

where $\longrightarrow = \big\{\big(p, \alpha, X(p), p\big) : p \in d, \alpha \in Act\big\}$.

Each state $p \in 2^F$ of the coordinator has a single self-loop with a role annotation corresponding to the feature combination p. The composition of the coordinator with the RBA thus enforces that exactly those roles are played that correspond to the selected feature combination. Analogous to the feature-controller formalism presented in Dubslaff et al. (2015), modeling dynamic product lines can be achieved by adding appropriate transitions between the coordinator states.

Having a transformation from fMDPs to RBAs raises the questions whether the reverse transformation is also possible. Assuming that the RBA does not contain any overriding transitions, that is indeed the case as evidenced by the following definition. The main idea of the transformation is to treat role annotations as feature guards which arise from the conjunction of the annotation's literals.

Definition 14 (*fMDP Induced by An RBA*)**.** The *corresponding fMDP of an RBA* $\mathcal{A} = (S, Act, \langle R, \varnothing\rangle, \longrightarrow, S^{init})$ is defined as

$$\text{Fmdp}[\mathcal{A}] \ = \ \big(S,\ R,\ Act,\ \text{P},\ S^{init}\big)$$

where $\text{P} = \big\{\big(s, \bigwedge_{a \in X} a, \alpha, \lambda\big) : s \xrightarrow{\alpha/X} \lambda\big\}$.

Since the transformed RBA must not contain any unbound roles, the transformation is only applicable to RBAs that result from the composition of the system's individual RBAs, i.e., after all roles have been bound. The transformation of RBAs that represent roles is not possible as overriding transitions and the role-binding operator have no equivalent within fMDPs. Thus, the transformation cannot be used to derive compositional feature-oriented systems where role behaviors are represented in feature modules but only towards monolithic annotative family models of the whole role-based system (Dubslaff, 2021).

4. Implementation

Based on the theoretical foundations presented in the previous section, we have implemented a tool that enables formal modeling and analysis of role-oriented systems. We have designed a role-oriented modeling language that allows a succinct description of RBAs and their composition as well as coordination. The MDP-semantics of an RBA under a coordinator allows us to apply a translational analysis approach. A role-oriented model is translated into the input language of the probabilistic model checker Prism, which subsequently carries out the analysis. This approach is similar to the approach of the feature-oriented analysis tool ProFeat (Chrszon et al., 2018) that has shown to be beneficial for the quantitative analysis of feature-oriented systems and relies on a translation of fMDP specifications into Prism input.

In the following, we give an overview of the modeling language and provide notable details of the translation approach. To

```
1 system {
2   acc[2] : Account;
3   c : Checking; src : Source; trg : Target;
4   trans : MoneyTransfer;
5
6   c boundto acc[0]; src boundto c;
7   trg boundto acc[1];
8   src in trans; trg in trans;
9 }
```

List. 10. System description for the banking example.

illustrate important language features, we rely on the banking example from the previous sections. In addition to the natural types and roles as introduced earlier, the model code will contain a checking role, allowing to overdraw the bank account when the account is empty, and a compartment type for the money transfer. For more information on the modeling language and the implementation, we refer to Chrszon (2021).

4.1. Role-oriented modeling language

A role-oriented model consists of two distinct parts: a specification of the system's structure, i.e., a role model, and behavioral definitions for all components. The distinction between these two descriptions is inspired by modeling languages for feature-oriented systems (e.g. Classen et al., 2012; Chrszon et al., 2018; Dubslaff, 2019), 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 (Kühn et al., 2015), RBAs defining component behaviors are given through a guarded command language (Jifeng et al., 1997; Alur and Henzinger, 1999) that is similar to the one used to describe models for the probabilistic model checker PRISM.

4.1.1. System structure

System instances are defined by a set of constraints that list the components of the system and specifies how they are related to each other. In particular, a constraint may state that a role is bound to a given player and that a role is contained in a specific compartment. The constraint language is based on first-order logic and interpreted over the set of component instances. An example is shown in Listing 10. In lines 2 to 4, the component instances are listed, where the constraint in line 2 defines an array containing two accounts. The role binding is specified using the `boundto` keyword as shown in lines 6 and 7. Similarly, the `in` keyword is used to define the membership of a role in a compartment (line 8). A set of constraints might be satisfied by multiple systems. In this case, the `system` block defines a system family rather than a single system. An example for this is shown in Listing 11. The first two lines state that there are two accounts and a money transfer compartment, but no roles are defined or bound. The constraint in line 5 then specifies that there should be a role (either `Source` or `Target`) bound to each of the accounts and that these roles must be contained in the `trans` compartment. There are two system instances fulfilling this constraint set, corresponding to a money transfer from the first account to the second and vice versa. Defining system families this way is a convenient approach to verify the functional correctness of each variant of a system. In case of a quantitative analysis, the optimal configuration, e.g., the optimal assignment of roles to players, w.r.t. some measure can be synthesized.

```
1 system {
2   acc[2] : Account;
3   trans : MoneyTransfer;
4
5   forall i : [0..1]. exists r : role.
6     r boundto acc[i] & r in trans;
7 }
```

List. 11. Description of a system family.

```
1 natural type Account;
2 role type Checking(Account);
3 role type Source(Account, Checking);
4 role type Target(Account, Checking);
5 compartment type MoneyTransfer(Source, Target);
```

List. 12. Role model for the banking example.

4.1.2. Type level

While the RBA formalism concerns solely the instance level, the modeling language additionally provides a type level to allow instantiating components multiple times. Type definitions specify additional constraints that each system instance must satisfy. Listing 12 shows the type definitions for the banking example. Each role-type definition is followed by a list of component types whose instances are allowed as players of the role. Similarly, a compartment-type definition specifies the types of roles whose instances must be contained in an instance of the compartment. The set of type definitions can be seen as a textual representation of a role model, such as the CROM (Kühn et al., 2015), that describes the basic structure for every system instance.

4.1.3. Operational behavior

The definitions for the component behaviors are strictly separated from the type declarations and the system structure declarations. Operational behavior is defined within *modules* where each module corresponds to a single RBA. A component type can be associated with one or more modules which "implement" the type, as shown in line 1 of Listing 13. Here, the `bal_counter` module is given as implementation for the `Account` type. When a component type is instantiated, all its associated modules are instantiated as well. Associating modules to types rather than instances enables a generic modeling of component behavior. This allows us to easily extend a system by creating new instances without having to define the behavior of each new instance individually.

Modules are described using an extension of PRISM's guarded command language. A module may contain one or more variables that define the state space of the module and a set of guarded commands that describe the transitions between those 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 command will synchronize with commands of other components that are labeled with the same action. The module presented in Listing 13 implements a balance counter that can be increased and decreased using the respective actions. The `self` keyword is replaced with the component instance name upon instantiation, such that each account has distinct actions for changing the balance. For `MAX_BAL=1`, the RBA described the module corresponds to the account RBA shown in Fig. 4.

```
1 impl Account(bal_counter);
2
3 module bal_counter {
4   bal : [0..MAX_BAL] init 1;
5
6   [self.inc] bal < MAX_BAL -> (bal' = bal + 1);
7   [self.dec] bal > 0 -> (bal' = bal - 1);
8 }
```

List. 13. Behavior definition for the `Account` type.

4.1.4. Role-specific behavior

For defining the operational behavior of role components, the language provides additional constructs. We consider the module for the checking account role presented in Listing 14. The `override` keyword in front of the action label (lines 4 and 5) marks all corresponding transitions of the command as overriding, i.e., for a role instance c all transitions are annotated with $\{+c\}$. Furthermore, the `player` keyword is provided to refer to the component to which the role component is bound. Assuming that the `system` block contains the constraint `acc[0] boundto c`, then the `player` keyword is replaced by `acc[0]` upon instantiation of `c`. Note that the role module does not contain any explicit role annotations. The appropriate annotations are automatically added to each transition by the implementation. Additionally, the implementation automatically converts any role to a non-blocking role w.r.t. its player, such that the original behavior of the player is preserved in case the role is not played.

4.1.5. Coordination of role-playing

A model may contain one or more coordinators. A coordinator definition is similar to a module, but allows for additional coordination commands. Coordinator commands have the form `[action] [role-guard] guard -> update`, where `role-guard` is a Boolean expression over role instances defined in the model. A coordinator command synchronizes with exactly those transitions of the system whose role annotation satisfies the role guard. Consider the example presented in Listing 15. The command in line 4 states that the checking role must be played to overdraw the account in case the account balance has reached zero. Conversely, the other command prohibits playing a checking role if there is still money left in the account. Note that we can easily define these rules for all roles of type `Checking` that exist in the model by using the `forall` keyword as shown in line 2.

4.1.6. Costs and rewards

To reason about quantitative properties such as energy consumption and throughput, the states and transitions of a model can be annotated with costs and rewards. As in the Prism language, transition rewards are associated with an action label. Additionally, a reward definition may include an additional role guard similar to coordination commands. Then, the reward is assigned to all transitions whose role-playing annotations satisfy the role guard. An example is shown in Listing 16 where a fee is applied to every `withdraw` action in case the corresponding source role is played. The `forall` keyword may be used within reward structures as well to define rewards for all components of certain types at once.

4.2. Translation into Prism

As described in Section 3, a (probabilistic) role-oriented system is constructed by composing the RBAs for the naturals, roles, and compartments, followed by a composition with the coordinator which ultimately results in a single MDP. Lifting this approach to the guarded command language outlined in Section 4.1 leads to a straightforward translation of the role-oriented modeling language into the Prism language. First, the component modules are composed, followed by a composition with the coordinator modules. The result is then a single module in the Prism language that encompasses the whole system behavior. However, applying the compositions directly on the module level almost always results in an exponential blowup in the number of guarded commands in the Prism model, which renders even the analysis of small to medium-sized models infeasible. Therefore, we propose an alternative translation approach. Formally, each RBA $\mathcal{A}$ is transformed into a separate MDP $\mathcal{M}[\mathcal{A}]$ with multi-actions. In a multi-action MDP (maMDP), transitions are labeled with sets of actions rather than single actions. This transformation encodes role-playing annotations of an RBA into the action sets of the corresponding maMDP. To mimic binding a role r in an RBA $\mathcal{R}$ to some player $\mathcal{M}$ within maMDPs, we define a closure operation $cl^r_{\mathcal{R}}(\mathcal{M})$ that further transforms the player's corresponding maMDP. The maMDPs obtained from the transformation of all RBAs can then be composed via parallel composition, resulting again in a single MDP representing the whole system. The transformation can be lifted to the guarded command language which allows us to translate a role-oriented model into a Prism model with multi-actions while preserving its compositional structure. Since the number of guarded commands only grows polynomially during the transformation, the exponential blowup of the model representation is avoided. We have previously implemented an extended version of Prism that can directly handle models with multi-actions (Baier et al., 2018) and thus performs the parallel composition of the multi-action modules internally. In the following, a formal definition for the alternative translation approach and its correctness is provided in detail.

4.2.1. Transformation of RBAs to maMDPs

We provide a formal definition of the transformation from RBAs to multi-action MDPs. Given a set Y of role annotations, we define $Y^- = \{r : +r \in Y\}$. Further, we define the set of overriding actions of a role instance r in an RBA $\mathcal{R} = (S, Act, R, \longrightarrow, S^{init})$ as $Act^{+r}_{\mathcal{R}} = \{\alpha : (s, \alpha, X, \lambda) \in \longrightarrow, +r \in X\}$ where $r \in R$.

Definition 15 (*maMDP of an RBA*)**.** The *multi-action MDP arising from an RBA* $\mathcal{A} = (S, Act, R, \longrightarrow, S^{init})$ is defined as

$$\mathcal{M}[\mathcal{A}] = \left(S, Act \cup \mathbb{A}(R), \longrightarrow_{\mathcal{M}}, S^{init}\right)$$

where $\longrightarrow_{\mathcal{M}}$ is the smallest transition relation fulfilling the following rule:

$$\frac{s \xrightarrow{\alpha/X} \lambda}{s \xrightarrow{\{\alpha\} \cup X \cup X^-}_{\mathcal{M}} \lambda}$$

The above definition is sufficient to mimic the parallel composition of RBAs by a parallel composition of the transformed maMDPs. However, it does not cover the effects of role binding, in particular the (add) rule in Fig. 6 which needs special treatment. Suppose the RBA of role r has a transition annotated with $+r$ and labeled with action α which is shared by the role and its player. Since the action α is shared, the transformed maMDPs of the role and the player will potentially synchronize over α. In this case, we must ensure that the player's maMDP remains in the same state (according to the intended semantics of $+r$). To achieve this effect, a self-loop labeled with $\{\alpha, +r\}$ is added to every state of the player's maMDP. Since no other transition of the player is labeled with an action set Σ where $+r \in \Sigma$, the player remains in the same state in case of $+r$ annotated transitions. This transformation is formalized by the following definition.

```
impl Checking {
  od : [0..MAX_OVERDRAFT] init 0;

  [override player.inc] od > 0 -> (od' = od - 1);
  [override player.dec] od < MAX_OVERDRAFT -> (od' = od + 1);
}
```

List. 14. Behavior definition for the `Checking` role type.

```
coordinator {
  forall c : Checking {
    [player(c).dec] [!c] player(c).bal > 0 -> true;
    [player(c).dec] [c] player(c).bal = 0 -> true;
  }
}
```

List. 15. Coordinator specifying that `Checking` roles can only be played if the account balance has reached 0.

```
rewards "fees" {
  forall src : Source {
    [src.withdraw] [src] true := 0.05;
  }
}
```

Listing 16. Annotation of costs on transitions.

Definition 16 (*RBA Closure*)**.** Given an RBA $\mathcal{R}$ and an instance name r, the *closure* over a multi-action MDP $\mathcal{M} = (S, Act, \longrightarrow, S^{init})$ is defined as the multi-action MDP

$$cl^r_{\mathcal{R}}(\mathcal{M}) = \left(S,\ Act,\ \longrightarrow',\ S^{init}\right)$$

where $\longrightarrow' = \longrightarrow \cup \left\{(s, \{\alpha, +r\}, Dirac(s)) : s \in S, \alpha \in Act^{+r}_{\mathcal{R}}\right\}$.

The compatibility of role-binding and the parallel composition of RBAs with the parallel composition of the transformed maMDPs is given by the following theorem. Note that the concept of non-synchronizing actions in the parallel composition of maMDPs (cf. Definition 4) is needed here to distinguish between role-playing actions and "standard" actions which are synchronizing and non-synchronizing, respectively.

Theorem 2 (*Composition Compatibility*)**.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be RBAs with compatible role interfaces, Act_1, Act_2 be the action sets of $\mathcal{A}_1, \mathcal{A}_2$ respectively, $\mathcal{R}$ be an RBA for role r that can be bound to $\mathcal{A}_1$ and $\cong$ denote equivalence up to isomorphism. Then with $N = Act_1 \cup Act_2$*

1. $\mathcal{M}\left[\mathcal{A}_1 \| \mathcal{A}_2\right] \cong \mathcal{M}[\mathcal{A}_1] \parallel_N \mathcal{M}[\mathcal{A}_2]$
2. $\mathcal{M}\left[\mathcal{R}[r \to \mathcal{A}_1]\right] \cong cl^r_{\mathcal{R}}\left(\mathcal{M}[\mathcal{A}_1]\right) \parallel_N \mathcal{M}[\mathcal{R}]$

Proof. We first show (1). Let $\mathcal{A}_i = (S_i, Act_i, R_i, \longrightarrow_i, S^{init}_i)$ for $i \in \{1, 2\}$. Further, let $\mathcal{M}_j = (S_{\mathcal{M}_j}, Act_{\mathcal{M}_j}, \longrightarrow_{\mathcal{M}_j}, S^{init}_{\mathcal{M}_j})$ for $j \in \{1, 2, 21, 22\}$ be multi-action MDPs with $\mathcal{M}_1 = \mathcal{M}\left[\mathcal{A}_1 \| \mathcal{A}_2\right]$, $\mathcal{M}_{21} = \mathcal{M}[\mathcal{A}_1]$, $\mathcal{M}_{22} = \mathcal{M}[\mathcal{A}_2]$ and $\mathcal{M}_2 = \mathcal{M}_{21} \parallel_N \mathcal{M}_{22}$. We show that $\mathcal{M}_1 \cong \mathcal{M}_2$. $S_{\mathcal{M}_1} = S_{\mathcal{M}_2}$, $Act_{\mathcal{M}_1} = Act_{\mathcal{M}_2}$ and $S^{init}_{\mathcal{M}_1} = S^{init}_{\mathcal{M}_2}$ follows directly from Definitions 6, 15 and 4. It remains to show that $\longrightarrow_{\mathcal{M}_1} = \longrightarrow_{\mathcal{M}_2}$.

Ad "$\subseteq$": The transitions of $\mathcal{M}_1$ arise from the parallel composition of the two RBAs $\mathcal{A}_1, \mathcal{A}_2$ defined by the rules in Definition 6 which we consider separately by case distinction.

(int$_1$) Assume there is a transition $s_1 \xrightarrow{\alpha_1/X}_1 \lambda_1$ with $\alpha_1 \in Act_1 \setminus Act_2$. Then, by applying rule (int$_1$) of Definitions 6 and 15, it follows that for all $s_2 \in S_2$ the MDP $\mathcal{M}_1$ has transitions

$$\langle s_1, s_2\rangle \xrightarrow{\{\alpha_1\} \cup X \cup X^-}_{\mathcal{M}_1} \lambda_1 * Dirac(s_2)\ .$$

On the right hand side, the transformation of Definition 15 applied on $\mathcal{A}_1$ yields the transition $s_1 \xrightarrow{\{\alpha_1\} \cup X \cup X^-}_{\mathcal{M}_{21}} \lambda_1$. We apply rule (int$_1$) in Fig. 3 which yields for all $s_2 \in S_2$ the transitions

$$\langle s_1, s_2\rangle \xrightarrow{\{\alpha_1\} \cup X \cup X^-}_{\mathcal{M}_2} \lambda_1 * Dirac(s_2)\ .$$

(int$_2$) The symmetric case follows from case (int$_1$) by swapping $\mathcal{A}_1$ and $\mathcal{A}_2$.

(sync) Assume there are transitions $s_1 \xrightarrow{\alpha_1/X}_1 \lambda_1$ and $s_2 \xrightarrow{\alpha_2/Y}_2 \lambda_2$ with $\alpha = \alpha_1 = \alpha_2$. Then, by applying rule (sync) of Definitions 6 and 15 we obtain that $\mathcal{M}_1$ has a transition

$$\langle s_1, s_2\rangle \xrightarrow{\{\alpha\} \cup (X \cup Y) \cup (X \cup Y)^-}_{\mathcal{M}_1} \lambda_1 * \lambda_2\ .$$

On the right hand side, the transformation of Definition 15 applied on $\mathcal{A}_1$ and $\mathcal{A}_2$ yield transitions $s_1 \xrightarrow{\{\alpha\} \cup X \cup X^-}_{\mathcal{M}_{21}} \lambda_1$ and $s_2 \xrightarrow{\{\alpha\} \cup Y \cup Y^-}_{\mathcal{M}_{22}} \lambda_2$. Rule (sync) of Fig. 3 applies, since $|\{\alpha\} \cup X \cup X^- \cup Y \cup Y^-) \cap N| = 1$ where $\{\alpha\} \subseteq N$. Note that $X \cap N = \varnothing$ and $Y \cap N = \varnothing$. This yields the transition

$$\langle s_1, s_2\rangle \xrightarrow{\{\alpha\} \cup X \cup X^- \cup Y \cup Y^-}_{\mathcal{M}_2} \lambda_1 * \lambda_2\ ,$$

equal to the transition in $\mathcal{M}_1$ since $(X \cup Y)^- = X^- \cup Y^-$.

Ad "$\supseteq$": This direction can be shown analogously to ($\subseteq$). Now let us show (2). Let $\mathcal{A}_p = (S_p, Act_p, R_p, \longrightarrow_p, S^{init}_p)$ be an RBA and $\mathcal{R} = (S_a, Act_a, R_a, \longrightarrow_a, S^{init}_a)$

with $\mathcal{A}_p$ and $\mathcal{R}$ having compatible role interfaces. Further, let $\mathcal{M}_j = (S_{\mathcal{M}_j}, Act_{\mathcal{M}_j}, \longrightarrow_{\mathcal{M}_j}, S^{init}_{\mathcal{M}_j})$ for $j \in \{1, 2, 21, 22\}$ be multi-action MDPs with $\mathcal{M}_1 = \mathcal{M}\left[\mathcal{R}[r \to \mathcal{A}_p]\right]$, $\mathcal{M}_{21} = cl^r_{\mathcal{R}}\left(\mathcal{M}[\mathcal{A}_p]\right)$, $\mathcal{M}_{22} = \mathcal{M}[\mathcal{R}]$ $\mathcal{M}_2 = \mathcal{M}_{21} \parallel_N \mathcal{M}_{22}$. We show that under a projection π that removes all +-annotated actions, $\mathcal{M}_1 = \pi(\mathcal{M}_2)$. $S_{\mathcal{M}_1} = S_{\mathcal{M}_2}$, $Act_{\mathcal{M}_1} = Act_{\mathcal{M}_2}$ and $S^{init}_{\mathcal{M}_1} = S^{init}_{\mathcal{M}_2}$ follows directly from Definitions 7, 15, 16 and 4. It remains to show that $\longrightarrow_{\mathcal{M}_1} = \pi(\longrightarrow_{\mathcal{M}_2})$.

Ad "$\subseteq$": The transitions of $\mathcal{M}_1$ arise from binding role r in $\mathcal{R}$ to $\mathcal{A}_p$ as defined by the rules in Fig. 6 which we consider separately by case distinction.

(int$_p$) Assume there is a transition $s_p \xrightarrow{\alpha_p/Y}_p \lambda_p$ with $\alpha_p \in Act_p \setminus Act_a$. Then, by applying rule (int$_p$) of Definitions 7 and 15,

it follows that for all $s_a \in S_a$ the MDP $\mathcal{M}_1$ has transitions

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha_p\}\cup Y\cup Y^-}_{\mathcal{M}_1} Dirac(s_a) * \lambda_p \ .$$

On the right hand side, the transformation of Definition 15 yields the transition $s_a \xrightarrow{\{\alpha_a\}\cup X\cup X^-}_{\mathcal{M}_{21}} \lambda_a$. We apply rule ($\text{int}_2$) of Fig. 3, which for all $s_a \in S_a$ yields the transitions

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha_p\}\cup Y\cup Y^-}_{\mathcal{M}_2} Dirac(s_a) * \lambda_p \ .$$

(int_a) This case can be shown similarly to (int_p).

(sync) Assume there are transition $s_p \xrightarrow{\alpha_p/X}_p \lambda_p$ and $s_a \xrightarrow{\alpha_a/Y}_{\mathcal{R}} \lambda_a$ with $\alpha = \alpha_p = \alpha_a$ and $+r \notin X$. Then, by applying rule (sync) of Definitions 7 and 15, it follows that $\mathcal{M}_1$ has a transition

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha\}\cup(X\cup Y)\cup(X\cup Y)^-}_{\mathcal{M}_1} \lambda_a * \lambda_p \ .$$

On the right hand side, the transformation of Definition 15 yields the transitions $s_a \xrightarrow{\{\alpha\}\cup X\cup X^-}_{\mathcal{M}_{21}} \lambda_1 * Dirac(s_p)$ and $s_p \xrightarrow{\{\alpha\}\cup Y\cup Y^-}_{\mathcal{M}_{22}} \lambda_p$. Since $(\{\alpha\}\cup X\cup X^-)\cap Act_{\mathcal{M}_{22}} = (\{\alpha\}\cup Y\cup Y^-)\cap Act_{\mathcal{M}_{21}} = \{\alpha\}$ and $|\{\alpha\}\cup X\cup X^-\cup Y\cup Y^-)\cap N| = 1$, rule (sync) in Fig. 3 applies. Note that $X\cap N = \varnothing$ and $Y\cap N = \varnothing$. This yields the transition

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha\}\cup X\cup X^-\cup Y\cup Y^-}_{\mathcal{M}_2} \lambda_a * \lambda_p \ .$$

(add) Assume there is a transition $s_a \xrightarrow{\alpha_p/X}_a \lambda_a$ where $\alpha \in Act_a \cap Act_p$ and $+r \in X$. Then, by applying rule (add) of Definitions 7 and 15, it follows that for all $s_p \in S_p$ the MDP $\mathcal{M}_1$ has transitions

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha_p\}\cup(X\setminus\{+r\})\cup X^-}_{\mathcal{M}_1} \lambda_a * Dirac(s_p) \ .$$

On the right hand side, $\mathcal{M}_{21}$ has transitions $s_p \xrightarrow{\{\alpha_p, +r\}}_{\mathcal{M}_{21}} Dirac(s_p)$ for all $s_p \in S_p$ introduced by the closure as defined in Definition 16. The transformation of Definition 15 applied on $\mathcal{R}$ yields that $\mathcal{M}_{22}$ has a transition $s_a \xrightarrow{\{\alpha_p\}\cup X\cup X^-}_{\mathcal{M}_{22}} \lambda_a$. Applying rule (sync) of Fig. 3 which is possible since $+r \in X$, yields for all $s_p \in S_p$ the transitions

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha, +r\}\cup X\cup X^-}_{\mathcal{M}_2} \lambda_a * Dirac(s_p) \ .$$

Removing all $+r$ actions using projection π then yields the transitions

$$\langle s_a, s_p \rangle \xrightarrow{\{\alpha\}\cup(X\setminus\{+r\})\cup X^-}_{\mathcal{M}_2} \lambda_a * Dirac(s_p) \ .$$

Ad "$\supseteq$": This direction can be shown analogously to ($\subseteq$). □

Having shown that the composition operators on RBAs are compatible with the parallel composition of their corresponding maMDPs, it is left to show that the transformation is correct in the sense that the MDP semantics of an RBA corresponds to the maMDP semantics of the transformed RBAs. For this, we first define the composition of a transformed RBA and a transformed role-playing coordinator, i.e., we define the $\bowtie$ operator from Definition 10 for maMDPs. To streamline the definition, we define the projection functions $act_R(\Sigma) = \Sigma \setminus \mathbb{A}(R)$ and $role_R(\Sigma) = \Sigma \cap \mathbb{A}(R)$.

Definition 17 (*Composition with Coordinator For maMDPs*)**.** The maMDP arising from the composition of an maMDP $\mathcal{M}_a = (S_a, Act_a, \longrightarrow_a, S_a^{init})$ representing a system and an maMDP $\mathcal{M}_c = (\mathcal{A}_c, Act_c, \longrightarrow_c, S_c^{init})$ of a coordinator for a set of roles R is defined as

$$\mathcal{M}_a \bowtie_R \mathcal{M}_c = \left(S_a \times S_c, Act_a \cup Act_c, \longrightarrow, S_a^{init} \times S_c^{init}\right)$$

where $\longrightarrow$ is the smallest transition relation fulfilling the rules in Fig. 17.

We now show the correctness of our translational approach that is used within our implementation. Note that the following theorem directly implies behavioral equivalence for RBAs without role-playing restrictions, i.e., with respect to the trivial coordinator comprising a single state and empty transition relation. First, we define a technical notion of *projections on multi-actions* to relate multi-action MDPs of a certain form with standard MDPs. For a given set of actions Act and roles R, let $\pi : 2^{Act\cup\mathbb{A}(R)} \to Act \times 2^R$ be a function that transforms a multi-action into a pair of action and role-playing where for any $\alpha \in Act_c \cup Act_a$ and $\mathcal{I} \subseteq R$ we have $\pi(\{\alpha\} \cup \mathcal{I}) = \langle \alpha, \mathcal{I} \rangle$. When $\mathcal{M}$ is an maMDP where in each transition at most one element is not from R, we write $\pi(\mathcal{M})$ to denote the MDP arising from the application of π onto every transition.

Theorem 3 (*Correctness*)**.** *For any RBA $\mathcal{A}$ and coordinator C with roles R_a and R_c, respectively, we have*

$$\pi\left(\mathcal{M}[\mathcal{A}] \bowtie_R \mathcal{M}[\mathsf{C}]\right) \quad \cong \quad [\![\mathcal{A}]\!]_{\mathsf{C}}$$

where $R = R_a \cup R_c$.

Proof. Let $\mathcal{A} = (S_a, Act_a, R_a, \longrightarrow_a, S_a^{init})$ be an RBA and $\mathsf{C} = (S_c, Act_c, R_c, \longrightarrow_c, S_c^{init})$ be a role-playing coordinator. Further, let $\mathcal{M}_1$ be an maMDP with $\mathcal{M}_1 = \mathcal{M}[\mathcal{A}] \bowtie_R \mathcal{M}[\mathsf{C}]$ and $\mathcal{M}_2$ be an MDP with $\mathcal{M}_2 = [\![\mathcal{A}]\!]_{\mathsf{C}}$, and $R = R_a \cup R_c$. Note that projection π as defined above can be applied to all multi-actions in $\mathcal{M}_1$, since multi-actions in $\mathcal{M}[\mathcal{A}]$ and $\mathcal{M}[\mathsf{C}]$ contain at most one action α with $\alpha \in Act_a$ and $\alpha \in Act_c$, respectively. Furthermore, the (sync) rule of the composition operator $\bowtie_R$ only produces multi-actions which contain at most one action $\alpha \in Act_a \cup Act_c$ (cf. Definition 17). We now show that $\pi(\mathcal{M}_1) = \mathcal{M}_2$. First, $S_1 = S_2 = S_a \times S_c$ and $S_1^{init} = S_2^{init}$ is clear by Definitions 15 and 17. Applying Definitions 15 and 17 yields $Act_1 = (Act_a \cup \mathbb{A}(R_a)) \cup (Act_c \cup \mathbb{A}(R_c)) = Act_a \cup Act_c \cup \mathbb{A}(R)$ and Definition 10 yields $Act_2 = (Act_a \cup Act_c) \times 2^R$, with $R = R_a \cup R_c$. Thus, we have $\pi(Act_1) = Act_2$. Finally, $\pi(\longrightarrow_1) = \longrightarrow_2$ is clear by the definition of $\bowtie_R$ (cf. Definition 17) which matches Definition 10. □

The transformation from RBAs to maMDPs outlined in this section can be lifted straightforwardly to guarded commands which allows us to translate a role-oriented model into the Prism language with multi-actions. It remains to discuss the translation of the coordinator which will be addressed in the next section.

4.2.2. Translation of the coordinator

The translation of coordinator commands needs to handle the additional role guard which must be resolved to every possible role-playing satisfying the guard. Assuming that a coordinator controls n distinct role instances, then each coordinator command may correspond to 2^n translated commands in the worst case. This may result in a prohibitively large translated coordinator even for small to medium-sized models. However, we can mitigate the exponential blowup based on the following two observations. First, when translating a coordinator, the set of role instances can be restricted to the set of roles that are actually appearing within its role guards. Second, usually not all roles appear in every role guard. In fact, most coordinator commands focus on small groups of roles, e.g., those that are contained in the same compartment, or even on single roles. We can therefore apply an optimized translation that first separates the coordinator into smaller localized coordinators. 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

$$(\mathrm{int}_a)\ \frac{s_a \xrightarrow{\Sigma}_a \lambda_a \qquad act_R(\Sigma) \cap act_R(Act_c) = \varnothing \qquad \mathcal{I} \in \mathbb{R}(role_R(\Sigma), R)}{\langle s_a, s_c \rangle \xrightarrow{act_R(\Sigma) \cup \mathcal{I}} \lambda_a * Dirac(s_c)}$$

$$(\mathrm{int}_c)\ \frac{s_c \xrightarrow{\Sigma}_c \lambda_c \qquad act_R(\Sigma) \cap act_R(Act_a) = \varnothing \qquad \mathcal{I} \in \mathbb{R}(role_R(\Sigma), R)}{\langle s_a, s_c \rangle \xrightarrow{act_R(\Sigma) \cup \mathcal{I}} Dirac(s_a) * \lambda_c}$$

$$(\mathrm{sync})\ \frac{\begin{array}{c} s_a \xrightarrow{\Sigma_a}_a \lambda_a \qquad s_c \xrightarrow{\Sigma_c}_c \lambda_c \\ act_R(\Sigma_a) \cap act_R(Act_2) = act_R(\Sigma_c) \cap act_R(Act_1) \\ |act_R(\Sigma_a \cup \Sigma_c)| = 1 \qquad \mathcal{I} \in \mathbb{R}(role_R(\Sigma_a \cup \Sigma_c), R) \end{array}}{\langle s_a, s_c \rangle \xrightarrow{act_R(\Sigma_a \cup \Sigma_c) \cup \mathcal{I}} \lambda_a * \lambda_c}$$

Fig. 17. Rules for the composition of an maMDP arising from an RBA and an maMDP arising from a coordinator.

for each partition is generated. Each of these modules coordinates a smaller set of roles 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.

4.2.3. Encoding role-playing into states

In an optional step of the translation described previously, the resulting Prism model can be transformed such that the role-playings are encoded into state labels (De Nicola and Vaandrager, 1990). This allows for expressing and analyzing role-playing properties with standard Prism property specifications.

5. Evaluation

To evaluate our implementation and the whole approach of role-oriented modeling and analysis, we conducted several experiments and selected three experimental studies to answer the following research questions:

- (**RQ**$_1$) Is the translational approach of turning role-oriented models into Prism input language and models effective and scalable?
- (**RQ**$_2$) Can our approach be used to detect and pinpoint role interactions?
- (**RQ**$_3$) Are RBAs eligible to model context- and self-adaptive behaviors and to be used to synthesize optimizing adaptation strategies?
- (**RQ**$_4$) To what extent role-oriented approaches are beneficial compared to feature-oriented approaches?

For all the illustrative examples, the actual analysis is performed on automatically translated role-oriented models using Prism in its multi-action MDP supporting version (Baier et al., 2018). The implementation and the experiments are provided in the artifact (Anon, 2022) accompanied to this article. We run the experiments on a system with two quad-core Intel Xeon L5630 CPUs (at 2.13 GHz) and 192 GB RAM running Debian 10.

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 helps to positively answer RQ$_1$, RQ$_2$, and RQ$_4$. The second example deals with a self-adaptive robot production cell, focusing on quantitative effects of interactions, mainly to answer RQ$_3$ but also supporting RQ$_2$. The third experiment issues an elevator product line, implementing a classical community benchmark (Plath and Ryan, 2001) from the feature-oriented domain in our role-oriented modeling language. This example mainly contributes to RQ$_4$.

5.1. Peer-to-peer file transfer

Our first example issues a peer-to-peer file transfer scenario, inspired from Hennicker and Klarl (2014) and already mentioned in the introduction to motivate hierarchical interactionss. 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. We specified the model using the role-oriented input language defined in Section 4.1 and then translated into the standard Prism language (see Section 4.2 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.

5.1.1. 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 c has just sent a request and received the requested file, respectively. In

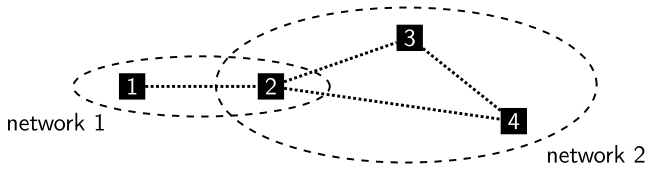


Fig. 18. Scenario with two overlapping networks.

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.[1]

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 Fig. 18, comprising 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 interactionss. 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 19 presents the action trace[2] 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 (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 10), those roles actively involved in interactions can be easily traced down and could help to resolve undesired behavior.

[1] Property (3) required a fair scheduling among the different stations.

[2] For better readability, the states have been omitted and the trace has been reformatted, but has not been altered otherwise.

Contributing to RQ_2, we conclude that role interactions can have crucial impact on behaviors of role-oriented systems, can be detected even at higher levels of compartments, and traced down by active interplays.

5.1.2. Interactions in feature-oriented systems

The functional analysis of the file-transfer example revealed that network–local coordination is not sufficient in case networks interact by means of shared stations. Fixing (undesired) hierarchical interactionss might hence require coordination across multiple networks. Here, the role-oriented view is beneficial and provides coordination capabilities also between multiple systems through compartments. Note that a system of feature-oriented systems might also be modeled using a purely feature-oriented approach, e.g., by joining the feature models of the individual systems. However, modeling shared stations (like station 2 in Fig. 18) is not immediately possible and modeling their behavior requires separate feature modules for each station in each network. Thus, to guarantee that the separate feature modules are actually representing a single station, they have to be synchronized, which is technically achieved via shared actions and parallel composition. However, these shared actions would also provide active interplays that have to be disregarded as they are actually not standing for interactions but are modeling remains. Here, role-oriented modeling provides a clean composition approach that accurately captures the structure of the system of systems.

We thus observe that hierarchical interactions do not have a corresponding counterpart in (plain) feature-oriented systems. In combination with the positive answer to RQ_2 of the last section, this supports our role-oriented approach to interactions.

Contributing to RQ_4, our role-oriented approach can detect interactions not detectable with feature-oriented approaches.

5.1.3. 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-oriented 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-oriented 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 (Klein et al., 2018). The time for translating the role-oriented models is not 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). An attempted analysis of a system with 8 stations and 3 files failed due to memory limitations. Fig. 20 visualizes the analysis time of the role-oriented 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 (Baier et al., 2018) uses a different encoding of actions

```
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
```

List. 19. Action trace for the violation of property (3).

Table 1
Model sizes, build times, and analysis times for the file-transfer models.

Stations	Files	States	Role-oriented model			Standard PRISM model		
			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	1	83	10 810	0.352	0.030	1 540	0.076	0.032
4	1	328	24 873	0.760	0.063	3 773	0.124	0.073
3	2	981	45 127	1.368	0.129	5 533	0.221	0.150
5	1	1 063	44 900	1.552	0.143	7 229	0.225	0.166
6	1	3 126	69 837	2.874	0.261	11 433	0.362	0.299
7	1	8 625	101 313	4.847	0.415	17 506	0.624	0.383
4	2	13 937	98 575	3.702	0.483	13 646	0.745	0.608
8	1	22 748	138 624	6.997	0.654	24 054	0.923	0.807
9	1	58 007	185 849	10.658	1.172	34 362	1.581	1.628
5	2	130 779	177 263	9.245	1.596	29 539	2.598	1.815
4	3	235 843	208 195	9.494	1.525	23 171	2.230	1.612
6	2	1 035 819	281 856	17.535	5.299	64 419	7.862	5.513
7	2	7 450 395	419 064	35.844	13.237	134 222	21.522	17.964
5	3	8 770 909	381 840	27.208	13.111	69 169	13.079	13.771
8	2	50 362 803	593 018	65.928	31.791	200 442	51.566	41.718
5	4	193 690 961	683 121	93.195	86.554	123 723	66.381	97.930
6	3	222 866 842	668 929	126.077	119.330	226 758	164.400	163.973
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

compared to standard PRISM, causing an additional overhead[3]. Even though, the benefits of our modular role-oriented modeling becomes apparent in larger instances where the overhead is superseded. For the largest instances with 2 or 3 files, the structure of the role-oriented model turned out to be favorable and lead to a faster analysis.

Our role-oriented analysis approach is effective and feasible even for large models, providing a positive answer to RQ_1.

5.2. Self-adaptive production cell

To address RQ_3, we consider an example from production automation by means of an automated production cell that adapts itself in case of failures (inspired by Güdemann et al. (2006)). 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 Güdemann et al. (2006) we applied a role-oriented 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. In order to enable the quantitative analysis of the system, we utilized the probabilistic version of our framework.

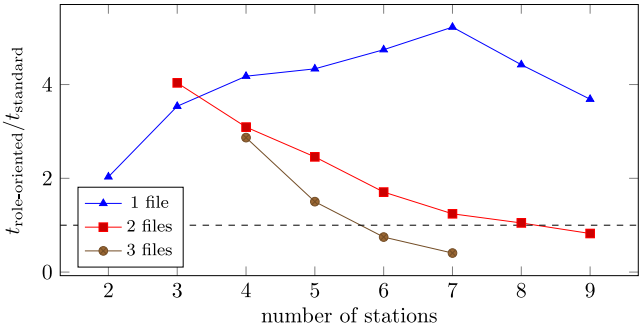


Fig. 20. Combined build and analysis time of role-oriented model relative to the standard model for fixed numbers of files.

[3] This overhead can surely be reduced by optimizing the prototypical implementation provided in Baier et al. (2018).

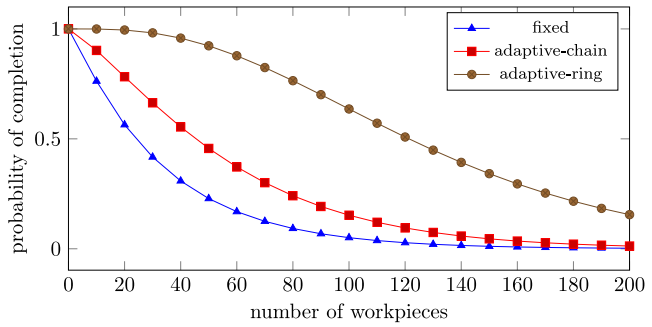


Fig. 21. Maximal probability of processing n workpieces.

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

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 Fig. 21. 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 Fig. 22. 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 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 Fig. 22. 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 interactionss. 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. Fig. 22). 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.

Contributing to RQ_2, our approach can quantify the impact of role interactions. For context- and self-adaptive systems, our role-oriented approach can be used to guide adaption strategies, providing a positive answer to RQ_3.

5.3. Elevator product line

As the third example, we consider an elevator system (Plath and Ryan, 2001), a classical product line that has been considered in various publications on family-based verification and analysis (e.g. Apel et al., 2013c; Cordy et al., 2013; Chrszon et al., 2018). Using this example, we demonstrate how the concept of features can be mapped to roles and how role-playing annotations aid in detecting feature interactions.

The elevator system consists of a cabin that travels between the floors of a multi-story building. It implements a "Single Button Collective Control" scheme, i.e., there is only a single button on each landing for calling the elevator without the ability to specify the desired direction. The elevator will serve all requests in one direction until no more requests in the current direction are pending, in which case the direction is reversed. This strategy guarantees that each request is served eventually.

Plath and Ryan (2001) developed a feature-oriented extension of NuSMV's modeling language SMV and applied it to extend the elevator system with five optional features. They subsequently analyzed the elevator family using the NuSMV model checker and detected several feature interactions. Based on the NuSMV model, we have recreated the elevator system in our role-oriented modeling language (RML). As SMV and RML differ in their structure and semantics, several changes where necessary when adapting the elevator model. In SMV, the different components of the elevator system communicate over shared variables and move synchronously, i.e., all components jointly update their local state within a single transition. Furthermore, features are integrated using *superimposition* which directly modifies the model's SMV code. In the RML model, on the other hand, the components communicate and synchronize over shared actions. In particular, the elevator controller module controls the cabin by synchronization over the actions `open`, `close`, `up`, and `down`. For each feature in the original SMV model, the RML model contains a corresponding role. The integration of a feature is then accomplished by binding the corresponding role to the elevator controller, possibly overriding some of its actions and thus changing the system's behavior. An example is shown in Listing 23. In line 2, the role for feature `ttf` (two-thirds full) is instantiated and bound to the controller. Next, the `parking`

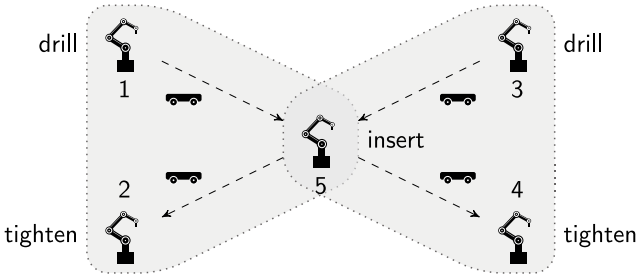


Fig. 22. Two automated production cells sharing a robot.

```
1  system {
2    ttf : TTF; ttf boundto control;
3    parking : Parking; parking boundto ttf;
4  }
5
6  coordinator {
7    []      [parking &  ttf] true -> true;
8    [open]  [parking & !ttf] true -> true;
9    [up]    [parking & !ttf] true -> true;
10   [down]  [parking & !ttf] true -> true;
11 }
12
```

List. 23. Instantiation of the elevator system with the two-thirds-full feature and the parking feature.

feature's corresponding role is instantiated and bound. Note that the `parking` role is not bound directly to the controller, but rather to the already bound `ttf` role. This allows the `parking` role to adapt both the behavior of `control` as well as `ttf`. Since bound roles are not necessarily played, we further have to define a coordinator that forces the roles to be played in order to activate the corresponding features' behavior. This is shown in lines 7–10. Here, we make use of role guards to give the `parking` role priority over the `ttf` feature.

Using our implementation (see Section 4) and Prism, we have checked whether the properties proposed in Plath and Ryan (2001) hold in order to detect feature interactions. The analysis was carried out for an elevator serving 3 floors. However, the model is configurable and a higher number of floors may be chosen We were able to reproduce the verification results reported by Plath and Ryan (2001), with the exception of some interactions that are caused by the specific encoding of the system and its properties in the SMV model and that are thus not present in the RML model. On the other hand, we found an additional interaction in the RML model. However, this interaction is caused by the specific encoding in the role-oriented model and thus it is neither present in the SMV model nor is it inherently contained in the elevator product line. Within RML models, overriding of behavior is only possible on actions, but not states. For this reason, some features must (partially) duplicate variables of the controller module. Specifically, the two-thirds-full feature has its own direction variable. In conjunction with the parking feature, this variable can become different from the controller's direction variable in some situations, which causes the cabin to cycle between two floors indefinitely and thus violates the guaranteed service property. We have found this reason for the interaction by replaying the counter-example trace on a system variant that does not show the unintended behavior. Together with the role annotations in the traces, this allowed us to pinpoint both the feature as well as the action that caused the interaction.

In conclusion, this example showed that embedding the feature concept is not only theoretically possible, but also supported by our modeling language. Furthermore, role annotations, which in this case correspond to feature activity, help finding the reasons for feature interactions.

For RQ_4, our role-oriented approach covers superimposition aspects also for feature-oriented systems, enabling superimposition also for FTSs and fMDPs, not implemented and evaluated till now.

5.4. Limitations and Threats to Validity

Since the presented role-oriented analysis approach is ultimately based on (probabilistic) model checking, it inherits also its limitations. This mainly concerns the scalability of the approach to large models. However, the translational approach we employ also provides the potential of applying well-known state-space explosion mitigation techniques and approaches such as statistical model checking.[4] Another limitation is the translation of the role-playing coordinator, which, in the worst case, can lead to an exponential blowup of the number of lines in the Prism model (as detailed in Section 4.2.2). This worst case can only occur if there is a coordinator command whose role-guard references most or all roles in the model. However, this was never the case in the analyzed models.

The internal validity of our analysis time measurements is threatened by multiple potentially interfering factors, such as the operating system, the Java virtual machine, and the hardware itself. In order to minimize interference, the analysis time has been averaged over three runs with an additional warm-up run beforehand and the number of other processes running on the system has been minimized. The results obtained for the overhead caused by the role-oriented analysis approach (see Section 5.1.3) may depend on the specific structure of the considered model, which threatens external validity. However, we are confident that the observed effect, i.e., diminishing overhead for a growing model size, also applies to other models. In fact, the overhead of the approach, though unquantified, did not prevent the analysis of the other considered models. To minimize the influence of the Prism models' structure on the symbolic representation within Prism, we applied reordering techniques for optimizing the variable ordering of the underlying data structure of MTBDDs (Klein et al., 2018).

6. Related work

While there are existing formal approaches to specify and validate *role-oriented* systems on the conceptual level (Kühn et al., 2015), support for analyzing their behavior is rather limited. In

[4] Statistical model checking is also supported by Prism and thus can be utilized without changes to the approach or the implementation.

the *Objects with Roles* (Pernici, 1990) approach, the role concept is employed to model the evolving behavior of objects. The behavior of objects as well as that of roles is described uniformly using state transition rules. A role may change the behavior of an object by adding or removing rules. The work only considers the behavioral aspect of roles and is only concerned with modeling and does not address any analysis. Roles have also been considered within architectural modeling (Allen and Garlan, 1997). Here, a role defines the expected behavior and the obligations of a system component. Thus, they are not utilized to capture context-dependent adaptations. To the best of our knowledge, the most closely related work to ours is the HELENA approach by Hennicker and Klarl (Hennicker and Klarl, 2014), where roles played by components collaborate within ensembles towards fulfilling a common goal. They define an operational semantics for HELENALIGHT in terms of transition systems (Hennicker et al., 2015; Klarl, 2015), which enables the verification using the SPIN model checker. Ensembles and components correspond to compartments and naturals in our approach, respectively. In HELENALIGHT, the components are passive in the sense that they serve as storage for data and provide operations which can be invoked by its roles. Since the components have no own behavior, it cannot be changed by its roles. This limited interaction between components and roles can easily be modeled in our approach with synchronization. Components in HELENALIGHT communicate asynchronously over (bounded) channels. Using synchronous communication, asynchronous communication can be modeled by explicitly modeling the channel components. The role concept has also received attention for designing and implementing multi-agent systems (MAS) (Cabri et al., 2010). Here, roles are used as an abstraction to capture the expected behavior of an agent in terms of capabilities, obligations, and requirements. As such, role based approaches within MAS are not compositional, as the agents must be already equipped with the required behavior. In Ren et al. (2006), the *Actors, Roles and Coordinators* (ARC) model, a decentralized role-based coordination model, is proposed. It includes a formal model for role behaviors. In contrast to our approach, roles in ARC coordinate a set of actors by modifying the messages sent between them. Several analysis approaches for *role-based access control* policies have been presented in the literature, e.g., Ahmed and Tripathi (2003), Rakkay and Boucheneb (2009), Zhang et al. (2005). In this context, roles are mainly associated with rights and bring no operational behavior themselves.

The adaptation mechanism of *aspect-oriented programming* (AOP) (Kiczales et al., 2001) is conceptually similar to that provided by roles. Thus, we will also consider formal approaches for AOP in the following. A modular model-checking approach for collaboration-based systems was presented in Fisler and Krishnamurthi (2001) and extended in Thang and Katayama (2003b,a). Here, the integration of extensions is realized by taking the union of the transition systems of extension and adapted component, and subsequently introducing additional transitions between them for switching between their behaviors. In our approach, this can be achieved by overriding some of the player's actions within the role. If the sequential execution of the role and player behavior is desired (in contrast to the parallel execution), the role may block all actions of the player when the role is played. A similar approach for verifying aspects that are inlined into the control-flow of a program has been presented in Krishnamurthi et al. (2004). Finally, in Sipma (2003) aspects for modular transition systems are considered. These systems are described using a first-order representation over variables, similar to a guarded command language. Aspects can modify the system by adding and removing behavior, similarly to roles in our approach.

Another conceptually related approach is *context-oriented programming* (COP) (Hirschfeld et al., 2008; Salvaneschi et al., 2012) which provides programming language constructs for dynamically adapting to context changes. Behavioral adaptations are grouped into layers which are activated or deactivated depending on the current context. Since COP is mainly considered from a programming language perspective, little attention has been given to formal modeling and analysis. In Kamina et al. (2011), an approach for verifying layer activation using model checking is presented. Our role-oriented approach may also be applicable for COP, if we consider a layer as a compartment and its contained behavioral adaptations as roles. Layer activation is then controlled by the role-playing coordinator.

7. Discussion and conclusion

Towards detecting hierarchical interactionss and active interplayss, we presented a compositional modeling approach that relies on concepts from role-oriented 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 interactionss and active interplayss using PRISM.

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 Chrszon et al. (2016). Other interesting directions are the development of a graphical notation for RBAs similar to UML statecharts (OMG, Object Management Group, 2011), explicit formal support for role-specific requirements, and the integration of our tool with existing modeling tools. Our role-oriented specification language might be also the starting point to implement transformations of superimposition on feature-oriented systems into PROFEAT, which would enable a formal analysis of existing real-world feature-oriented programming languages that mainly rely on superimposing feature modules (Schaefer et al., 2010; Apel et al., 2013a; Dubslaff, 2021).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Acher, M., Collet, P., Fleurey, F., Lahire, P., Moisan, S., Rigault, J.-P., 2009. Modeling context and dynamic adaptations with feature models. In: 4th International Workshop Models@run.time at Models 2009 (MRT'09). p. 10.

Ahmed, T., Tripathi, A.R., 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, pp. 196–203.

Allen, R., Garlan, D., 1997. A formal basis for architectural connection. ACM Trans. Softw. Eng. Methodol. 6 (3), 213–249. http://dx.doi.org/10.1145/258077.258078.

Alur, R., Henzinger, T.A., 1999. Reactive modules. Form. Methods Syst. Des. 15 (1), 7–48. http://dx.doi.org/10.1023/A:1008739929481.

Anon, 2022. Featuring roles: Artifact. URL https://github.com/pchrszon/rbsc.

Apel, S., Batory, D.S., Kästner, C., Saake, G., 2013a. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

Apel, S., Kästner, C., 2009. An overview of feature-oriented software development. J. Object Technol. 8, 49–84.

Apel, S., von Rhein, A., Thüm, T., Kästner, C., 2013b. Feature-interaction detection based on feature-based specifications. Comput. Netw. 57 (12), 2399–2409. http://dx.doi.org/10.1016/j.comnet.2013.02.025.

Apel, S., Rhein, A.v., Wendler, P., Groesslinger, A., Beyer, D., 2013c. Strategies for product-line verification: Case studies and experiments. In: Proceedings of the 2013 International Conference on Software Engineering. ICSE '13, IEEE Press, Piscataway, NJ, USA, pp. 482–491.

Bachman, C.W., Daya, M., 1977. The role concept in data models. In: Proceedings of the Third International Conference on Very Large Data Bases - Volume 3. VLDB '77, VLDB Endowment, pp. 464–476, URL http://dl.acm.org/citation.cfm?id=1286580.1286629.

Baier, C., 2005. Probabilistic models for reo connector circuits. J. UCS 11 (10), 1718–1748. http://dx.doi.org/10.3217/jucs-011-10-1718.

Baier, C., Chrszon, P., Dubslaff, C., Klein, J., Klüppelholz, S., 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. pp. 38–56.

Baier, C., Katoen, J.-P., 2008. Principles of Model Checking. MIT Press.

Cabri, G., Leonardi, L., Ferrari, L., Zambonelli, F., 2010. Role-based software agent interaction models: a survey. Knowl. Eng. Rev. 25 (04), 397–419.

Calder, M., Kolberg, M., Magill, E.H., Reiff-Marganiec, S., 2003. Feature interaction: a critical review and considered forecast. Comput. Netw. 41 (1), 115–141. http://dx.doi.org/10.1016/S1389-1286(02)00352-3, URL http://www.sciencedirect.com/science/article/pii/S1389128602003523.

Chandy, K.M., Misra, J., 1988. A Foundation of Parallel Program Design. Addison-Wesley, Reading, MA.

Chrszon, P., 2021. Formal analysis of variability-intensive and context-sensitive systems. (Ph.D. thesis). TU Dresden, URL https://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa2-736915.

Chrszon, P., Baier, C., Dubslaff, C., Klüppelholz, S., 2020. From features to roles. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume a - Volume a. SPLC '20, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3382025.3414962.

Chrszon, P., Dubslaff, C., Baier, C., Klein, J., Klüppelholz, S., 2016. Modeling role-based systems with exogenous coordination. In: Theory and Practice of Formal Methods. In: LNCS, Vol. 9660, Springer, pp. 122–139.

Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C., 2018. ProFeat: feature-oriented engineering for family-based probabilistic model checking. Form. Asp. Comput. 30 (1), 45–75.

Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.-Y., 2012. Model checking software product lines with SNIP. Int. J. Softw. Tools Technol. Transf. 14 (5), 589–612. http://dx.doi.org/10.1007/s10009-012-0234-1.

Classen, A., Cordy, M., Schobbens, P.-Y., Heymans, P., Legay, A., Raskin, J.-F., 2013. Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. IEEE Trans. Softw. Eng. 39 (8), 1069–1089.

Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns. Addison-Wesley Professional.

Coplien, J.O., Reenskaug, T., 2014. The DCI paradigm: Taking object orientation into the architecture world. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.), Agile Software Architecture. Morgan Kaufmann, Boston, pp. 25–62. http://dx.doi.org/10.1016/B978-0-12-407772-0.00002-2.

Cordy, M., Schobbens, P.-Y., Heymans, P., Legay, A., 2013. Beyond boolean product-line model checking: Dealing with feature attributes and multi-features. In: Proceedings of the 2013 International Conference on Software Engineering. ICSE '13, IEEE Press, Piscataway, NJ, USA, pp. 472–481.

De Nicola, R., Vaandrager, F., 1990. Action versus state based logics for transition systems. In: Guessarian, I. (Ed.), Proceedings of LITP Spring School on Theoretical Computer Science 1990. In: LNCS, Vol. 469, Springer, Berlin, Heidelberg, pp. 407–419.

Dehnert, C., Junges, S., Katoen, J., Volk, M., 2017. A storm is coming: A modern probabilistic model checker. In: Majumdar, R., Kuncak, V. (Eds.), Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II. In: Lecture Notes in Computer Science, Vol. 10427, Springer, pp. 592–600. http://dx.doi.org/10.1007/978-3-319-63390-9_31.

Dubslaff, C., 2019. Compositional feature-oriented systems. In: Ölveczky, P.C., Salaün, G. (Eds.), Software Engineering and Formal Methods. Springer International Publishing, Cham, pp. 162–180.

Dubslaff, C., 2021. Quantitative analysis of configurable and reconfigurable systems. (Ph.D. thesis). TU Dresden, Institute for Theoretical Computer Science.

Dubslaff, C., Baier, C., Klüppelholz, S., 2015. Probabilistic model checking for feature-oriented systems. Trans. Aspect-Oriented Softw. Dev. 12, 180–220.

Dubslaff, C., Koopmann, P., Turhan, A.-Y., 2019. Ontology-mediated probabilistic model checking. In: Proceedings of the 15th Conference on Integrated Formal Methods. IFM, LNCS:11918, Springer, pp. 194–211.

Dubslaff, C., Weis, K., Baier, C., Apel, S., 2022. Causality in configurable software systems. In: Proceedings of the 44th International Conference on Software Engineering. ICSE.

Fisler, K., Krishnamurthi, S., 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. pp. 152–163.

Garvin, B.J., Cohen, M.B., 2011. Feature interaction faults revisited: An exploratory study. In: Proceedings of the 22nd International Symposium on Software Reliability Engineering. ISSRE, ACM, pp. 90–99.

Gomaa, H., Hussein, M., 2003. Dynamic software reconfiguration in software product families. In: PFE. pp. 435–444.

Güdemann, M., Ortmeier, F., Reif, W., 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. pp. 38–47. http://dx.doi.org/10.1007/11839569_4.

Hennicker, R., Klarl, A., 2014. Foundations for ensemble modeling – the Helena approach. In: Iida, S., Meseguer, J., Ogata, K. (Eds.), Specification, Algebra, and Software. In: Lecture Notes in Computer Science, Vol. 8373, Springer Berlin Heidelberg, pp. 359–381.

Hennicker, R., Klarl, A., Wirsing, M., 2015. Model-checking Helena ensembles with SPIN. In: Logic, Rewriting, and Concurrency - Essays Dedicated to José Meseguer on the Occasion of His 65th Birthday. pp. 331–360.

Hirschfeld, R., Costanza, P., Nierstrasz, O., 2008. Context-oriented programming. J. Object Technol. 7 (3), 125–151. http://dx.doi.org/10.5381/jot.2008.7.3.a4.

Holl, G., Grünbacher, P., Rabiser, R., 2012. A systematic review and an expert survey on capabilities supporting multi product lines. Inf. Softw. Technol. 54 (8), 828–852. http://dx.doi.org/10.1016/j.infsof.2012.02.002.

Jifeng, H., Seidel, K., McIver, A., 1997. Probabilistic models for the guarded command language. Sci. Comput. Program. 28 (2), 171–192, Formal Specifications: Foundations, Methods, Tools and Applications.

Kamina, T., Aotani, T., Masuhara, H., 2011. Eventcj: a context-oriented programming language with declarative event-based context transition. In: Borba, P., Chiba, S. (Eds.), Proceedings of the 10th International Conference on Aspect-Oriented Software Development, AOSD 2011, Porto de Galinhas, Brazil, March 21-25, 2011. ACM, pp. 253–264. http://dx.doi.org/10.1145/1960275.1960305.

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., Carnegie-Mellon University Software Engineering Institute.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G., 2001. Getting started with ASPECTJ. Commun. ACM 44 (10), 59–65. http://dx.doi.org/10.1145/383845.383858.

Klarl, A., 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. pp. 39–45.

Klein, J., Baier, C., Chrszon, P., Daum, M., Dubslaff, C., Klüppelholz, S., Märcker, S., Müller, D., 2018. Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata. STTT 20 (2), 179–194.

Krishnamurthi, S., Fisler, K., Greenberg, M., 2004. Verifying aspect advice modularly. In: Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004, Newport Beach, CA, USA, October 31 - November 6, 2004. pp. 137–146. http://dx.doi.org/10.1145/1029894.1029916.

Kühn, T., Böhme, S., Götz, S., Aßmann, U., 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. pp. 113–124. http://dx.doi.org/10.1145/2814251.2814255.

Kühn, T., Leuthäuser, M., Götz, S., Seidl, C., Aßmann, U., 2014. A metamodel family for role-based modeling and programming languages. In: Software Language Engineering. Springer, pp. 141–160.

Kuhn, D.R., Wallace, D.R., Gallo, A.M., 2004. Software fault interactions and implications for software testing. IEEE Trans. Softw. Eng. 30, 418–421.

Kwiatkowska, M., Norman, G., Parker, D., 2002. PRISM: Probabilistic symbolic model checker. In: Computer Performance Evaluation: Modelling Techniques and Tools. Springer, pp. 200–204.

Legay, A., Delahaye, B., Bensalem, S., 2010. Statistical model checking: An overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (Eds.), Runtime Verification. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 122–135.

Mauro, J., Nieke, M., Seidl, C., Yu, I.C., 2016. Context aware reconfiguration in software product lines. In: Proceedings of the 10th Workshop on Variability Modelling of Software-Intensive Systems (VaMoS). ACM, pp. 41–48.

Mizoguchi, R., Kozaki, K., Kitamura, Y., 2012. Ontological analyses of roles. In: Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on. IEEE, pp. 489–496.

OMG, Object Management Group, 2011. Unified modeling language (UML): Superstructure version 2.4.1. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/.

Pernici, B., 1990. Objects with roles. SIGOIS Bull. 11 (2–3), 205–215. http://dx.doi.org/10.1145/91478.91542.

Plath, M., Ryan, M., 2001. Feature integration using a feature construct. Sci. Comput. Program. 41 (1), 53–84.

Post, H., Sinz, C., 2008. Configuration lifting: Verification meets software configuration. In: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy. IEEE Computer Society, pp. 347–350. http://dx.doi.org/10.1109/ASE.2008.45.

Puterman, M.L., 1994. Markov Decision Processes. Wiley, http://dx.doi.org/10.1002/9780470316887.

Qu, X., Cohen, M.B., Rothermel, G., 2008. Configuration-aware regression testing: An empirical study of sampling and prioritization. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis. ISSTA '08, Association for Computing Machinery, New York, NY, USA, pp. 75–86. http://dx.doi.org/10.1145/1390630.1390641.

Rakkay, H., Boucheneb, H., 2009. Security analysis of role based access control models using colored Petri nets and cpntools. In: Transactions on Computational Science IV. In: Lecture Notes in Computer Science, Vol. 5430, Springer Berlin Heidelberg, pp. 149–176. http://dx.doi.org/10.1007/978-3-642-01004-0_9.

Ren, S., Yu, Y., Chen, N., Marth, K., Poirot, P.-E., Shen, L., 2006. Actors, roles and coordinators — A coordination model for open distributed and embedded systems. In: Ciancarini, P., Wiklicky, H. (Eds.), Coordination Models and Languages. In: Lecture Notes in Computer Science, Vol. 4038, Springer Berlin Heidelberg, pp. 247–265.

Salvaneschi, G., Ghezzi, C., Pradella, M., 2012. Context-oriented programming: A software engineering perspective. J. Syst. Softw. 85 (8), 1801–1817. http://dx.doi.org/10.1016/j.jss.2012.03.024.

Schaefer, I., Bettini, L., Damiani, F., Tanzarella, N., 2010. Delta-oriented programming of software product lines. In: Proceedings of the 14th Conference on Software Product Lines. SPLC, LNCS:6287, Springer, pp. 77–91.

Schröter, R., Krieter, S., Thüm, T., Benduhn, F., Saake, G., 2016. Feature-model interfaces: the highway to compositional analyses of highly-configurable systems. In: Dillon, L.K., Visser, W., Williams, L.A. (Eds.), Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016. ACM, pp. 667–678. http://dx.doi.org/10.1145/2884781.2884823.

Sipma, H.B., 2003. A formal model for cross-cutting modular transition systems. In: Proc. of Foundations of Aspect Languages Workshop (FOAL03).

Steimann, F., 2000. On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng. 35 (1), 83–106.

Thang, N.T., Katayama, T., 2003a. 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. pp. 45–50.

Thang, N.T., Katayama, T., 2003b. Towards a sound modular model checking of collaboration-based software designs. In: 10th Asia-Pacific Software Engineering Conference (APSEC 2003), Chiang Mai, Thailand. pp. 88–97.

Trujillo-Tzanahua, G.I., Juárez-Martínez, U., Aguilar-Lasserre, A.A., Cortés-Verdín, M.K., 2018. Multiple software product lines: applications and challenges. In: Mejia, J., Muñoz, M., Rocha, A., Quiñonez, Y., Calvo-Manzano, J. (Eds.), Trends and Applications in Software Engineering. Springer International Publishing, Cham, pp. 117–126.

Vandin, A., ter Beek, M.H., Legay, A., Lluch Lafuente, A., 2018. Qflan: A tool for the quantitative analysis of highly reconfigurable systems. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (Eds.), Formal Methods. Springer International Publishing, Cham, pp. 329–337.

Yilmaz, C., Cohen, M.B., Porter, A.A., 2006. Covering arrays for efficient fault characterization in complex configuration spaces. IEEE Trans. Softw. Eng. 32 (1), 20–34.

Zave, P., 2001. Feature-oriented description, formal methods, and DFC. In: Gilmore, S., Ryan, M. (Eds.), Language Constructs for Describing Features. Springer London, London, pp. 11–26.

Zhang, N., Ryan, M., Guelev, D.P., 2005. Evaluating access control policies through model checking. In: Zhou, J., López, J., Deng, R.H., Bao, F. (Eds.), Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings. In: Lecture Notes in Computer Science, Vol. 3650, Springer, pp. 446–460. http://dx.doi.org/10.1007/11556992_32.