Modeling Role-Based Systems with Exogenous Coordination

Philipp Chrszon(✉), Clemens Dubslaff, Christel Baier, Joachim Klein, and Sascha Klüppelholz

Faculty of Computer Science, Technische Universität Dresden, Dresden, Germany
{chrszon,dubslaff,baier,klein,klueppel}@tcs.inf.tu-dresden.de

Abstract. The concept of roles is a promising approach to cope with context dependency and adaptivity of modern software systems. While roles have been investigated in conceptual modeling, programming languages and multi-agent systems, they have been given little consideration within component-based systems.

In this paper, we propose a hierarchical role-based approach for modeling relationships and collaborations between components. In particular, we consider the channel-based, exogenous coordination language Reo and discuss possible realizations of roles and related concepts. The static requirements on the binding of roles are modeled by rule sets expressed in many-sorted second-order logic and annotations on the Reo networks for role binding, context and collaborations, while Reo connectors are used to model the coordination of runtime role playing. The ideas presented in this paper may serve as a basis for the formalization and formal analysis of role-based software systems.

1 Introduction

Separation of concerns [19] is a well-established and accepted principle which appears in many modeling languages for computer systems. For instance, exogenous coordination languages such as Reo [2] aim at a clear separation of computational aspects and coordination (for a survey on coordination languages, cf. [38]). Within Reo, components encapsulate the operational behavior at the interface level and capture the computational aspects. For coordinating the components exogenously, a network of channels is used, which allows for any kind of synchronous and asynchronous communication. A further prominent example for separation of concerns is the distinction between entities and their relationships that is used, e.g., within the entity-relationship model for modeling relational databases. Entities and relationships are naturally complemented by the concept of roles [43]. Roles are often considered as placeholders in relationships and

The authors are supported by the DFG through the collaborative research centre HAEC (SFB 912), the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED and Institutional Strategy), the Research Training Groups QuantLA (GRK 1763) and RoSI (GRK 1907), and the DFG/NWO-project ROCKS, and Deutsche Telekom Stiftung.

E. Ábrahám et al. (Eds.): de Boer Festschrift, LNCS 9660, pp. 122–139, 2016.
DOI: 10.1007/978-3-319-30734-3_10

collaborations that are filled, i.e., played, by entities [41]. In this sense, roles are an abstraction of the expected behavior of the role player. They can be dynamically acquired and dropped by the role player, depending on the context and the relationships between their respective role players. Consider the example of a soccer player who participates in a local team and has the fortune to be also a player of the national team. In one team, he may play as a defender, while in the other team he plays as a midfielder. The respective role he acquires thus depends on the team he currently plays in, i.e., its local or national context. The team describes a collaboration by the roles of the team's players and additionally defines the relationships between the players through their roles as well. This justifies viewing roles as intermediaries between entities and relationships. Moreover, it promotes a further separation of concerns: roles encapsulate the varying aspects and behaviors of role players in different contexts. Therefore, roles improve the maintainability and extensibility of context-dependent systems with dynamically emerging collaborations between entities.

Although roles are intuitive and commonly understood, there is no generally accepted definition of roles [45]. Guarino and Welty stated in [23] that roles are such entities for which the ontological characterizations of *anti-rigidity* and *dependence* hold. Rigidity denotes that a property holds for an entity at all times and independently from the context, e.g., the property of being a person is rigid as it holds until the entity ceases to exist. The dual term anti-rigidity denotes properties that can cease to hold. For instance, a person can be a customer, but can also stop being a customer without ceasing to be a person. The second ontological notion of dependence describes entities whose existence depends on another entity, e.g., a customer is dependent on a vendor. Additionally, both customer and vendor depend on context, i.e., the exchange of money and goods. In the area of multi-agent systems and agent coordination, roles are widely regarded as an abstraction of behavior and are associated with a set of *requirements*, *capabilities* and *obligations* [13]. An agent must satisfy requirements in order to play a certain role and engages in collaborations with other agents according to its obligations. In object-oriented modeling and programming languages, the only commonly accepted trait of roles is that they can be played by unrelated objects [36,43]. For surveys about how roles can be further characterized, we refer to [36,43,45].

Towards modeling and designing role-based systems, several approaches have been proposed in the literature. In the Agent-Group-Role (AGR) meta-model for organizations in multi-agent systems [21,22], related agents play roles in groups. The group manager, which is a special role, coordinates role acquisition and removal of other actors. The role-oriented programming environment ROPE [11] uses a coordination language derived from Petri nets. The BRAIN proposal supports the analysis, design and implementation of role-based agent systems [14]. Within this approach, the notion of agent evolution as a central concept is introduced [12]. In the Actor-Role-Coordinator (ARC) approach, an agent system is divided in three distinct layers consisting of agents, roles and coordinators [42]. Roles are used as an abstraction of agent behavior and coordinate a group of

agents by means of message manipulation. Fasli presented a multi-modal logic framework based on the BDI paradigm (beliefs, desires, intentions) [20]. For a survey of role-based agent interaction models, see [13]. An operational model for role-based systems following the so-called *Helena approach* was presented in [24], where components play roles in groups to collaborate towards a goal. Their formal model combines relational and context-dependent roles and allows the application of model-checking techniques to reason about, e.g., reachability of the collaboration goal.

In this paper, we adopt the notion of roles that is commonly found in conceptual modeling (cf., e.g., [43]). Particularly, we rely on the meta-model for roles by Kühn et al. [35]. There, the concept of *compartments* is introduced, which captures both the collaborative and context-dependent nature of roles. Many role-based approaches consider either the collaborative or context-dependent aspect of roles, but not their combination [36,43]. As roles constitute intermediaries between entities and their relationships, and can be played by their assigned role players depending on their contexts, the coordination of roles is a central point when modeling role-based systems. Thus, it is rather natural to employ specialized coordination languages to describe role playing. However, coordination languages with roles and contexts have been given little consideration in the literature. For instance, [44] compared Reo with the ARC model and the Russian Reflective Dolls (RRD) approaches but focuses mainly on the expressivity of coordination languages, rather than on role-based systems.

The major goal of this paper is to provide first steps towards a theory of role-based exogenous coordination principles. For this, we rely on the channel-based coordination language Reo and show how to embed role-specific concepts. While previous work on role-based coordination mainly deals with monolithic approaches annotating role-playing agents, our framework is compositional and introduces *roles components* that might have their own behavior and are linked to their players via networks of channels that orchestrate the role-playing mechanisms. Role components can be bound to *atomic components* and *compartments*. Atomic components are standing for basic objects without incorporating role-based behaviors. Compartments are formalized by sets of role components, capable to formalize relationships or collaborations. By a set of rules expressed with logical formulas, we define static constraints on possible role bindings. We show that (as compartments describe *sets* of roles) many-sorted second-order logic provides an appropriate formalism with useful applications. Based on the logical characterization of role binding, the actual binding is modeled using coordination glue code in the form of Reo connectors to connect role players with role components and perform exogenous coordination to guarantee correct role playing. By annotating the modeled Reo network with compartments and role bindings, an organizational view of the system is induced.

The embedding of our formalism in Reo allows the application of the full machinery that has been developed for Reo also in the scope of role-based systems. In particular, the formal semantics for Reo [4,10,15,26] facilitate formal analysis and verification. Izadi et al. [25] introduced model checking techniques

for Reo networks using compositional reduction and abstraction techniques. In [16], Clarke presented a temporal logic and model-checking techniques for Reo networks with dynamic reconfigurations. The tool Vereofy [7,8] enables the verification of Reo connectors by means of model checking. Pourvatan et al. [39] provided an analysis technique based on symbolic executions of Reo networks. Kokash et al. [32,33] developed mappings from semantic formalisms for Reo to the process algebraic specification language of mCRL2 to enable data flow analysis in Reo networks. Proença and Clarke [40] presented data abstraction techniques for Reo networks. To reason about quantitative properties, Reo networks have been extended with timing constraints [5] and stochastic annotations [6,37]. These existing formalisms and tools for formal analysis and verification provide a well-grounded foundation that allow reasoning about role-based properties.

Outline: After a short primer on the exogenous coordination language Reo in Sects. 2 and 3 presents our framework for modeling role-based systems in Reo. There, we start with the building blocks to model roles, then illustrate role playing of atomic components and compartments and end with the formal framework on role binding. In Sect. 4, we discuss the application of formal analysis techniques and further research raised by our new framework.

2 A Short Primer on Reo Networks

We provide here a brief, high-level overview of the main concepts of Reo as well as the graphical representations we use in this paper for depicting Reo networks. For further details we refer to [2,10]. A Reo network, also called Reo circuit, is built from *components*, *channels* and *nodes*. In general, components serve to encapsulate operational behavior and can interact with the rest of the network via one or more interface ports (depicted as ○). Keeping to the spirit of exogenous coordination, with the coordination glue code between the components being formed by the Reo network, components generally do not know and need not be concerned about the environment in which they are used. Various semantics for Reo networks have been considered in the literature (see, e.g., [4,10,15,26]). On an intuitive level, it makes sense to think in terms of tokens that can be created, propagated, duplicated and consumed by the various parts of the network and that might optionally carry additional information (data). Channels in Reo have two channel ends and provide a rich variety of ways in which the activity at their incoming and outgoing ends can be related. Numerous channel types are predefined and the user may additionally provide customized channel semantics as needed. One of the most basic channels is the *synchronous channel* ⟶, which atomically propagates a token from its incoming end to its outgoing end. In contrast, the *FIFO1* channel —□→ can consume a token at its incoming end, store it in a single buffer cell, and can then propagate the token (and its data) later on via its outgoing end. Channels do not have to be unidirectional. For example, the *synchronous drain* channel →← has two incoming ends and may only consume tokens at both ends simultaneously. In contrast, the *asynchronous drain* →‖← can only consume a token at exactly

one of its ends at the same time. Another important channel is, e.g., the filter channel, whose behavior depends on the concrete data attached to a token. In case that the data matches the corresponding filter condition, the filter channel behaves as a synchronous channel and blocks otherwise. All these channels can serve as the basic building blocks for ensuring synchronicity and asynchronicity between different parts of the Reo network. The channel ends can connect to the interface ports of components as well as to Reo nodes, which serve to coordinate the flow of tokens between the connected channel ends. The standard Reo node (depicted as •) exhibits a *merger-replicator* semantics for passing tokens from the connected channel ends: Simultaneously, on the input side, it will nondeterministically choose exactly one of the available tokens (merge semantics). On the output side, it will propagate the token to all connected channel ends (replication semantics), duplicating as necessary. Crucially, a token can only be passed on if all the channel ends connected on the output side of the node are willing to consume the token simultaneously. This behavior allows for the elegant synchronization of an arbitrary number of connected channel ends. A variant of the standard node is the *router node* (depicted as ⊗), which retains the merger semantics on the input side but, in each step, propagates the token to exactly one of the connected channel ends on the output side.

From these basic ingredients, Reo networks representing a wide variety of interaction and coordination patterns can be built. A Reo network that provides such coordination is called a connector. This clear separation of computation inside components and coordination between components allows the construction of systems from reusable and easily exchangeable components and connectors. However, as the number of components of the system grows, Reo networks become increasingly complex. The ability to hierarchically encapsulate Reo networks into new components that can be used as building blocks on higher levels enables convenient separation of concerns and eases the design process of Reo networks. For this, the internal behavior of the Reo network is hidden from a higher-level Reo network perspective and interface ports are defined to allow for coordination of the new constructed component using a hiding mechanism.

Constraint automata [10] provide a compositional operational semantics for Reo, which enables standard verification techniques developed for labeled transition systems (see, e.g., [7,8,30,31,33]). This includes verification both of the coordination patterns in a network and – whenever automata-based specifications of the components' behavior are available – of the whole system.

3 Modeling Roles and Relationships in Reo

We introduce the concept of roles in the exogenous coordination language Reo to provide a methodology for constructing component-based systems. The Reo components we consider can be arbitrary Reo networks with interface ports arising, e.g., from hiding internal behaviors. On top of the Reo components used in the Reo network to model the system, also the role-based view on the network is organized in a hierarchical fashion. However, in contrast to the hiding operator applied to Reo networks to constitute components, we annotate role-specific

information to the components itself, providing a role-based modeling hierarchy orthogonal to the component-based modeling hierarchy of the Reo coordination language. The basic building blocks of this hierarchy are *atomic components*, *role components* and *compartments*. Atomic components stand for basic entities or agents which do not contain any role-specific behaviors, e.g., a person, a computer participating in a network, or a daemon run by an operating system. Role components encapsulate role-specific behaviors which enhance capabilities of entities, e.g., a soccer player role which can be played by a person, a server role which can be played by a computer within a network, or a scheduling role enhancing a daemon functionality. Compartments are built by collections of roles, i.e., by annotating sets of role components. They model relationships, collaborations and contexts of entities playing certain roles. For instance, following the examples mentioned above for atomic components and role components, a soccer team defines the context some soccer players are playing in, a file transfer compartment models the protocol to exchange data between servers and clients, or a desktop compartment coordinates the execution of daemons on desktop computers (which differs from the daemon coordination on server platforms). Compartments itself may serve as entities which also can play roles. Thus, the annotation of Reo components as atomic or the annotation of sets of role components as compartments, which both can be bound to roles, forms a role-based hierarchy of the Reo network. Obviously, the binding process of roles goes along with the modeling of the role-based system and should follow certain rules. For instance, a male person would in general not be bound to a daughter or mother role. We express such constraints within logical formulas in many-sorted second-order logic over possible annotations to the role-based Reo network.

In the following, we describe our approach for modeling role-based Reo networks in detail. For this, we follow the steps usually undertaken while modeling a role-based system within our framework. First, we introduce the building blocks of role components and how they are bound to (atomic) components. Then, we describe the role hierarchy established through compartments and their roles. Although the logical formulas for the rules of role binding on atomic components and compartments are usually fixed at the beginning of the modeling process (and possibly refined slightly during the construction of the Reo network), we introduce the formal framework for these rules at the end of this section. The reason for choosing this order is twofold. On the one hand, we would like to stepwise introduce the ingredients required to model role-based Reo networks in the order which forms the modeling hierarchy. For the formal framework of constraints on role bindings, however, all these ingredients have to be assumed as given. On the other hand, the formal framework for role-binding constraints does not require a Reo network with role-based annotations and can be used for more general purposes, not relying on Reo as coordination language. Thus, we describe the role-binding constraints separately from the Reo network modeling.

3.1 Representation of Roles

Components differ conceptually from both agents and objects. We consider the notion of components as an abstraction of behavior with a well

defined interface [1]. Indeed, a component can represent both an agent or an object or even sets thereof. Adopting the notion of roles as sets of requirements and obligations, roles could be modeled directly in the component-based framework of Reo by incorporating all role behaviors in Reo components. However, this approach leads to monolithic components that combine both their own and their roles' behavior. Hence, following this approach, there is no direct separation of concerns between behaviors of agents and objects and their capabilities enacted by the specific roles they play. Thus, modeling agents requires the knowledge of all roles and their behavior beforehand, which is in contrast to a compositional modeling approach. We thus propose a different approach where roles are considered as first-class entities that are represented and implemented by components. This approach has several important implications. Since a role instance is effectively a component, it can have its own state and behavior. Therefore, the role-specific behavior is encapsulated and not distributed over numerous components. Furthermore, this means a role can be played by several unrelated components. In our framework, a role adapts a component to a specific context, enabling the component to collaborate with other components in the same context. For example, a role component can implement a special communication protocol. The idea of adapting components by role-playing fits into the exogenous coordination model of Reo, because a component does not need to "know" the roles it plays. Furthermore, this approach eases reasoning about role-based systems as basic behaviors from agents and behaviors arising by role playing are separated and can be distinguished already during the modeling process.

3.2 Role Binding and Role Playing

To describe the concept of role binding and role playing incorporated into our framework, we first consider the basic case where both the role player and the role itself are modeled by Reo components. Before an atomic component can play certain roles, the roles must be bound to the component. The binding is realized by creating a Reo connector between the role components and the atomic component. This binding connector serves two tasks: It enables the atomic component to play the bound roles, and it realizes the coordination between the role components and the atomic component.

Every role comes with a set of capabilities and requirements. The capabilities gained by playing a role are encapsulated in the role component. A role component may provide additional ports, which equips the atomic component with additional means for communication. Obviously, a role component can only be bound to certain atomic components. These requirements are reflected in the set of ports the atomic component must provide and the set of rules specified in second-order logic over role names which we define in the last part of this section.

For the construction of the role binding connector, the full Reo framework can be employed. Thus, the binding connector may be arbitrarily complex and can implement various means of interaction and coordination between the atomic component and its role components. In the following illustrations, we depict atomic components as standard Reo components and role components with

rounded corners. Role binding is indicated by a dotted area around one atomic component and its bound roles.

Depending on how a role adapts an atomic component, different connectors may be appropriate. If a role component only adds additional behaviors, a connection realized by a standard Reo node is sufficient, as shown in Fig. 1 (a). Here, the output of the component $\mathcal{C}$ and role component $\mathcal{R}$ is merged, but $\mathcal{R}$ can neither modify nor block the output of $\mathcal{C}$ (assuming fairness). In the opposite case, the role binding removes or suppresses certain behavior of the atomic component. Here, the role component acts as a filter that only lets pass certain output data. This functionality can be realized by the connector shown in (b). The synchronous drain channel forces the synchronization between $\mathcal{C}$ and $\mathcal{R}$ for every outgoing message. Thus, if $\mathcal{R}$ refuses to synchronize, $\mathcal{C}$ cannot complete its send operation. Since the synchronous drain consumes both incoming messages, $\mathcal{R}$ the role component cannot add behavior by forging additional output data. In the most general case, a bound role component may suppress or modify any output of the atomic component and can create output data on its own as well. This is realized by the connector shown in (c), where all output data of $\mathcal{C}$ flows through $\mathcal{R}$.

Obviously, the binding pattern shown in (c) subsumes both (a) and (b), making them to seem redundant. However, the fact that the role component in (a) cannot modify or suppress output data is apparent in the binding connector, while in (c) one would have to examine the implementation of the role component to establish the same guarantee. Thus, by using behavior-restricting binding connectors, certain guarantees can be established without taking the components' concrete implementations into account. This also illustrates the compositional modeling approach for role-based systems where the coordination between roles is visible from the glue code between the role components and the playing atomic component.

While in Fig. 1(a) to (c) only one role is bound, a binding connector may also bind more than one role component to an atomic component, as exemplified in Fig. 1(d). Furthermore, the binding can introduce additional interface ports. The Reo networks $\mathcal{N}_1$ and $\mathcal{N}_2$ between the atomic component and the role components coordinate the role playing. Depending on the desired behavior, different connectors may be used in place of $\mathcal{N}_1$ and $\mathcal{N}_2$. For example, placing a router node ($\otimes$) between $\mathcal{C}$ and $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_n$ ensures that only exactly one role can be played at any given time. Contrarily, by using a standard Reo node ($\bullet$) for $\mathcal{N}_1$ all bound roles must be played at the same time. The network $\mathcal{N}_2$ determines the output behavior. For instance, if a standard Reo node is used, the output of exactly one role component is selected nondeterministically and sent to all output ports. Certainly, more complex connectors may be used for $\mathcal{N}_2$, such as a connector that merges the output data of the role components by creating tuples.

To illustrate the binding of role components to an atomic component in more detail, we turn to our first running example that shows a concrete implementation of the pattern presented in Fig. 1(d).

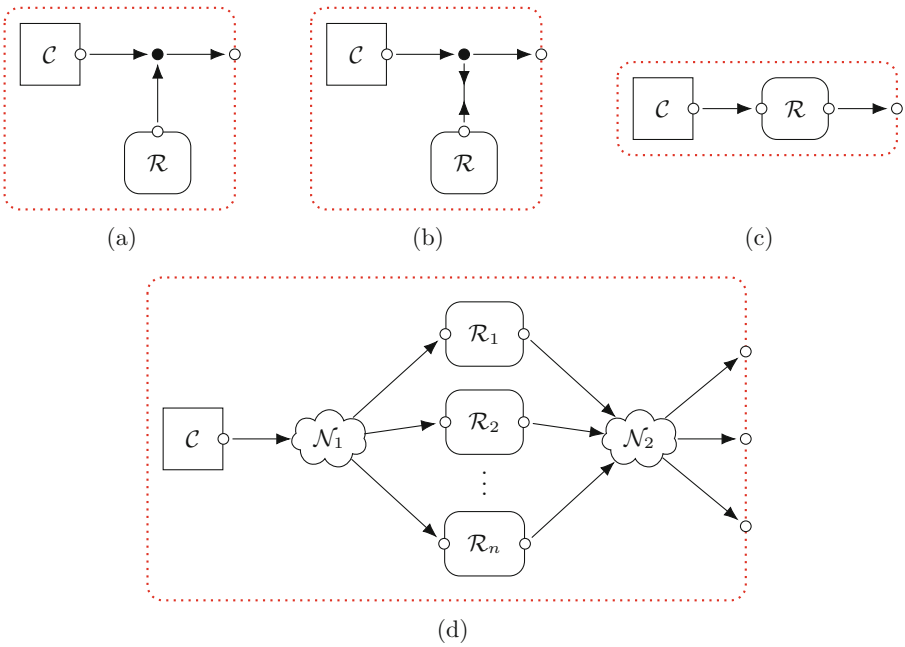

Fig. 1. Patterns for binding role components to an atomic component

Example 1. Figure 2 depicts a wine vending machine. The role components "white" and "red" add the capability to serve white wine and red wine, respectively. This is modeled by a pattern similar to Fig. 1(d) with two roles. For a formal semantics of the depicted Reo network, a compatible formalism to describe its components is required. In Fig. 3 we show two constraint automata [10], one capturing the operational behavior of the vending machine (without wine serving capabilities) and one for each wine role component. These constraint automata combined with the depicted Reo network directly yield the formal semantics for the vending machine, i.e., a constraint automaton modeling the whole behavior of the wine vending machine. Without any role playing, the wine vending machine dispenses drinks directly when it received the payment. After binding the white and red-wine role, the respective wine is only dispensed after the selection port is activated through exogenous coordination.

Until now, we only considered role binding which is a prerequisite for role playing. In our approach, a role is actively played if the behavior of its role component is observable, i.e., whenever one or more of its ports are active. Clearly, not all possible combinations of active roles are useful or valid. In our example of a wine vending machine, which is able to serve both red and white wine, it should not occur that both roles are played at the same time, i.e., eventually serving ros? wine. Thus, a binding connector not only connects an atomic components and its role components, but also coordinates role playing.

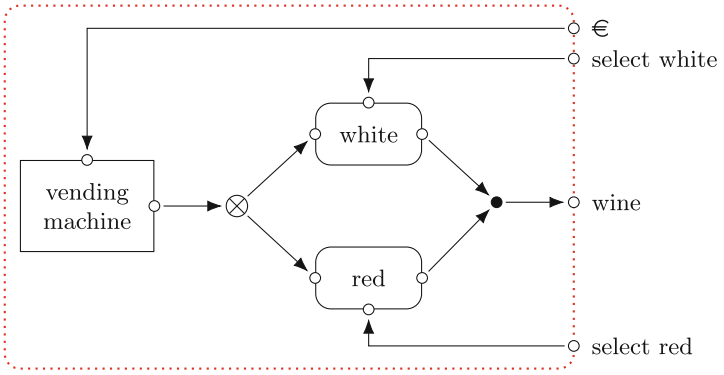

Fig. 2. Role binding connector of a wine vending machine

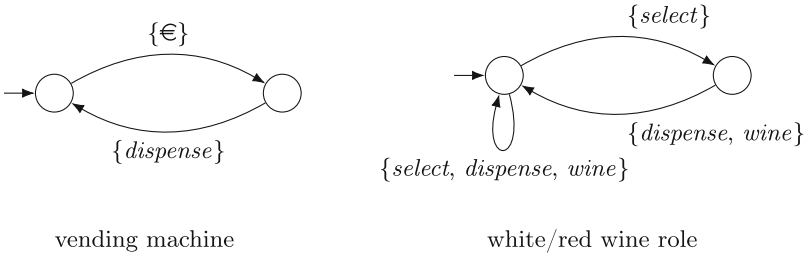

Fig. 3. Constraint automata for the vending machine and one wine role

In our wine vending machine, the merger semantics of the Reo node (on the right) ensures that only one role component can be active at the same time.

3.3 Role Relationships and Compartments

Roles often depend on one or more counter-roles [23]. Consider again Example 1 of a wine vending machine. Surely, a vending machine is only useful if there are customers that buy the goods it offers. Thus, both the *white* and *red* roles are dependent on a *white wine customer* and a *red wine customer*, respectively. While the previous section dealt with the relationship of role components and their role player, this section focuses on the relationships between role components.

Similar to role binding, role relationships are realized by Reo connectors between role components. The purpose of a role relationship connector is the coordination of role components, i.e., it influences and controls role playing. For instance, the connector depicted in Fig. 4 ensures that the vending machine plays the *white* role whenever the person plays the *white wine customer* role. As for role binding connectors, the coordination realized by the relationship connector can be arbitrarily complex. In our example, the connector has two purposes. First, it serves as a sequencer that allows for selecting wine only after money has been paid, modeled by the upper *FIFO1* channels (—▣→ depicts a filled

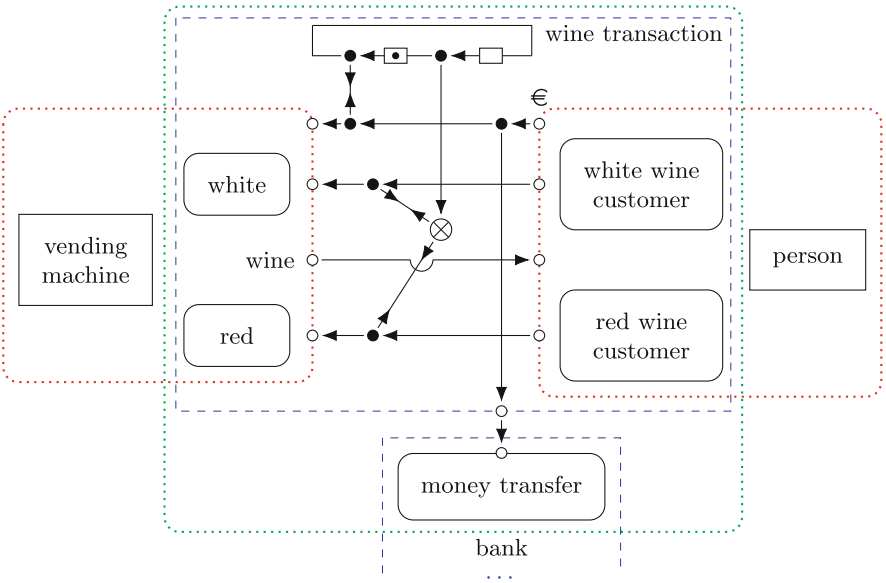

Fig. 4. Relationship connector in the wine transaction compartment

FIFO1 channel). Second, it disallows simultaneous selection of white and red wine, ensured by the *router node* in the center (depicted as ⊗).

Roles not only depend on relationships with other roles, but also on a context. Clearly, a person playing the *white wine customer* role cannot buy wine from a soda vending machine. Only in connection with a *white* wine role in a *wine transaction* context the *white wine customer* role can be played. We adopt the notion of *compartments* [36] as a representation of context and the collaboration of its roles. Compartments contain sets of role components and their role relationship connectors, depicted by a dashed rectangle surrounding their contained role components (see Fig. 4).

An important aspect of the wine *transaction* compartment is the payment. The customer may choose to pay using a credit card or paying cash. But then, the transaction itself plays the role of a *money transfer* in a *bank* compartment. Thus, not only atomic components, but also compartments themselves can play roles. Since compartments are sub-networks that may have external ports, the role binding approach presented in Sect. 3.2 can be applied to compartments as well.

As every role component is part of a compartment and every compartment can play roles itself, our modeling approach is hierarchical. Starting from atomic components as basic building blocks, role binding can be nested arbitrarily deep. Returning to our running example, the *bank* compartment itself may play the role of a *borrower* of another bank. Again, this bank can also play the role of a *borrower* of yet another bank, and so on.

3.4 Role-Based Reo Networks and Role-Binding Constraints

To formalize Reo networks modeling role-based systems, we use the concept of *types*. Types abstract away actual operational behavior and coordination to encapsulate the role-based view on parts of the Reo network. Every role component, atomic component or compartment as they appear during the modeling process illustrated in the last sections will be assigned a type. For what follows, let $\mathfrak{A}$ be the set of *atomic component types*, $\mathfrak{R}$ the set of *role component types*, and $\mathfrak{C}$ the set of *compartment types*. During the modeling process of role-based Reo networks, the types are usually known beforehand.

Definition 1. *A* role-based Reo network *is a tuple* $(\mathcal{N}, C_{atom}, C_{role}, \Delta, \beta, type_{atom}, type_{role}, type_{cprt})$ *where*

- $\mathcal{N}$ *is a Reo network over a set* C *of components,*
- *the set* C_{atom} *of atomic components and the set* C_{role} *of role components are disjoint subsets of* C*,*
- $\Delta \subseteq 2^{C_{role}}$ *is the set of compartments,*
- $\beta \colon C_{role} \to C_{atom} \cup \Delta$ *is a total function binding roles to atomic components or compartments,*
- $type_{atom} \colon C_{atom} \to \mathfrak{A}$, $type_{role} \colon C_{role} \to \mathfrak{R}$ *and* $type_{cprt} \colon \Delta \to \mathfrak{C}$ *annotate atomic components, role components and compartments with their type, respectively.*

In Example 1 we did not distinguish between types and their instances resulting in the role-based Reo network. Thus, the types can be assumed to agree with the instance names in this case. To illustrate how types are incorporated in role-based Reo networks, we chose an example from the soccer domain, which will serve as the running example for the rest of this section.

Example 2. Let $\mathfrak{A} = \{\mathsf{person}\}$, $\mathfrak{R} = \{\mathsf{keeper}, \mathsf{defender}, \mathsf{midfielder}, \mathsf{attacker}\}$, and $\mathfrak{C} = \{\mathsf{SoccerTeam}, \mathsf{Tournament}\}$ be types. Figure 5 depicts a part of a role-based Reo network, where the type assignment to some instance is done by captions of the form "instance : type". Frank as a person is capable of playing the role of a defender in both, his local and the national team. Whereas in the local team he plays with number 16 (as an instance of a player in the team), he has the role of number 7 in the national team. Edwin, as the second person we consider, can play the role of the keeper in the national team. Besides other players not depicted in Fig. 5, Frank and Edwin can take part in a world cup competition within the national team, modeled by a competitor role of the team instantiated as the second team in group B. We omitted the actual coordination networks as they concern only the role playing and do not appear within the annotations for role-based Reo networks. For instance, it can be assumed that the coordination network between Frank and its two defender roles models that he cannot play both roles simultaneously. Furthermore, the cloud in the national team compartment depicted in Fig. 5 stands for a coordination network and could also include state, e.g., that the color of the national team is orange.

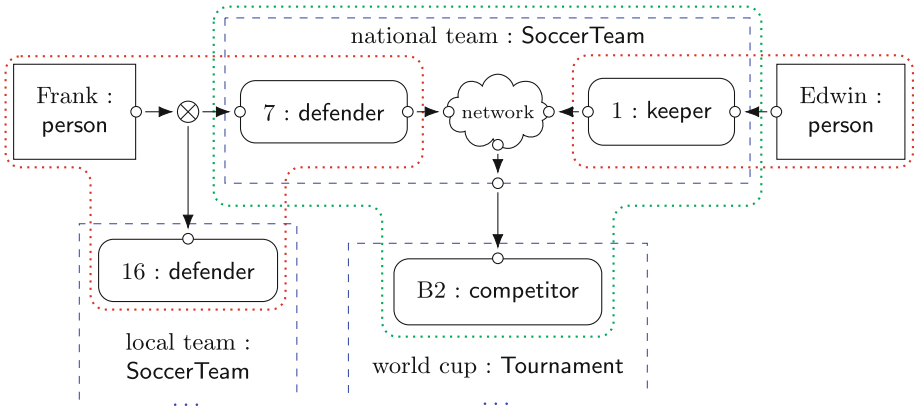

Fig. 5. Role-based Reo network of a soccer tournament

It is clear that a role-based Reo network should follow rules which guarantee consistency of the model according to the domain described. In the soccer domain for instance, it is nonsensical that a team attempts to play the role "keeper" in a competition. We express such rules which have to be obeyed for role-based Reo networks with formulas in many-sorted second-order logics (SOL). The sorts we distinguish here are the ones of role players (denoted *RolePlayer*) and role instances (denoted *RoleInst*). Atomic components and compartments are of sort *RolePlayer*, whereas role components are of the sort *RoleInst*. Role instances are assigned to role players by β of sort $\mathit{RoleInst} \to \mathit{RolePlayer}$. First-order variables range over atomic components and role components. For each atomic component type and role component type we identify its name with a predicate of arity one, evaluating to true if it is interpreted over an instance of that type. Similarly, a predicate for each compartment type is true if interpreted over a second-order variable containing all the role components of a compartment of that type. Set predicates (such as $\in$ and $\subseteq$) evaluate to false when applied on atomic components as role player instance. We call a set of sentences $\mathfrak{F}$ over the described SOL a *role rule set* and say that a role-based Reo network is *valid* if the network is a model for all sentences in $\mathfrak{F}$.

A common restriction on role binding (see, e.g., [24,35]) is the requirement that every role instance is part of at most one compartment. We do not enforce this restriction in our framework, e.g., to allow for modeling a father-son relation as a compartment contained in a family-relation compartment. However, this rule can be included into our role rule set as an SOL sentence

$$\forall \mathit{RoleInst}\ x, \mathit{RolePlayer}\ Y, Z.\ x \in Y \cap Z \ \Rightarrow\ Y = Z$$

Turning to our running example from the soccer domain, the rule that at least one keeper has to play in every soccer team can be expressed by the SOL sentence

$$\forall \mathit{RolePlayer}\ T.\ \mathsf{SoccerTeam}(T) \Rightarrow \exists \mathit{RoleInst}\ k.\ \mathsf{keeper}(k) \wedge k \in T$$

The restriction that every keeper role has to be played by a person is also useful:

$$\forall \mathit{RoleInst}\ k.\ \mathsf{keeper}(k) \Rightarrow \mathsf{person}(\beta(k))$$

It is easy to see that the role-based Reo network depicted in Fig. 5 is valid when the role rule set contains exactly the rules above. Usually, one fixes a role rule set according to the chosen domain and then models a valid role-based Reo network as described in the last two sections.

4 Conclusions and Future Work

We presented an approach on how the exogenous coordination language Reo can be used to model role-based systems. For this, we introduced a formal framework to express static requirements on the binding of roles to their role players, based on atomic component types, role component types and compartment types over which rules in many-sorted SOL are stated. In role-based Reo networks, instances of atomic component types and role component types correspond to concrete Reo components, whereas instances of compartment types include role components and a coordination network between them. Within our approach, the purpose of Reo is to model the coordination between roles and their players (e.g., to guarantee operational requirements on role playing) and between the roles in compartments. The latter also allows the coordination of the collaboration of roles taking place in compartments, an important feature of compartments not apparent within the "contexts" a role appears in. Obviously, modeling the coordination of roles remains a highly sophisticated task within our framework, where several conformance requirements are only given implicitly. For instance, although a soccer player can have a role in the local as well as the national team, he should not play both roles simultaneously. Thus, formal analysis of the operational behavior and the role playing over time is desirable to guarantee correctness of the role-base system model.

Formal Semantics and Analysis for Role-Based Reo Networks. There has been extensive research on formal semantics for Reo [26]. When the components as the building blocks of role-based Reo networks (e.g., atomic components or role components) are modeled using Reo compatible formalisms, we directly obtain a formal operational semantics, e.g., in terms of a constraint automaton. Such operational semantics captures all the modeled role-based behaviors and fulfills the static constraints on role binding provided by our framework. By introducing, e.g., additional port labels for ports which define whether a role is active or not, we can rely on standard analysis techniques to check conformance of role playing requirements. For instance, model-checking tools such as Vereofy [8] or mCRL2 [32,33] can then be used to check run-time requirements on role-based Reo networks. Such tools would allow for checking invariants on simultaneous role playing, e.g., whether roles are only played together with their counter-roles or whether in a soccer team at least seven but at most eleven players are acting.

Future Work. Several directions on how our approach can be extended are left for further work. On the modeling side, our framework currently supports only static role models without dynamic binding and unbinding. For this, research on rewriting operations already applied to Reo networks in [34] could be used in combination with methods ensuring that the role-based Reo network remains valid according to role rule sets. Concerning constraints on temporal aspects, the notion of role rule sets could be extended with temporal logic formulas, for which, however, a semantics on an operational model of Reo networks has to be developed in more detail. Also, an extension of role rule sets containing (contextualized) description logics [27] could be imagined. On the formal analysis side, algorithms to check many-sorted SOL requirements on role binding for role-based Reo networks could be investigated. Reasoning about the role rule sets itself, e.g., checking whether some rules are contradictory, requires specialized algorithms, possibly only applicable onto fragments of the logics we presented. An open field is also to incorporate annotations into the operational semantics of role-based Reo networks to reason about compatibility [17,18], e.g., whether the behavior of a player matches the roles' requirements and vice versa. In this spirit, also the formalization of collaboration goals [24] expressed for each compartment and their reachability during runtime could be investigated. Another aspect within our framework is controller synthesis [28–31] with respect to temporal requirements, e.g., as stated above. As usual, the coordination between components in component-based system modeling is the most difficult part. Thus, the modeling process of role-based Reo networks could heavily benefit from synthesized controllers serving as coordinating connectors between players and their bound roles, e.g., by synthesizing the Reo glue code [3,9].

References

1. Arbab, F.: Abstract behavior types: a foundation model for components and their composition. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2002. LNCS, vol. 2852, pp. 33–70. Springer, Heidelberg (2003)
2. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**, 329–366 (2004)
3. Arbab, F., Baier, C., de Boer, F.S., Rutten, J., Sirjani, M.: Synthesis of Reo circuits for implementation of component-connector automata specifications. In: Jacquet, J.-M., Picco, G.P. (eds.) COORDINATION 2005. LNCS, vol. 3454, pp. 236–251. Springer, Heidelberg (2005)
4. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2003. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003)
5. Arbab, F., Baier, C., de Boer, F.S., Rutten, J.M.M., Sirjani, M.: Models and temporal logical specifications for timed component connectors. Softw. Syst. Model. **6**(1), 59–82 (2007)
6. Arbab, F., Meng, S., Moon, Y.-J., Kwiatkowska, M.Z., Hongyang, Q.: Reo2MC: a tool chain for performance analysis of coordination models. In: 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 287–288. ACM (2009)

7. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S.: Formal verification for components and connectors. In: de Boer, F.S., Bonsangue, M.M., Madelaine, E. (eds.) FMCO 2008. LNCS, vol. 5751, pp. 82–101. Springer, Heidelberg (2009)
8. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S.: A uniform framework for modeling and verifying components and connectors. In: Field, J., Vasconcelos, V.T. (eds.) COORDINATION 2009. LNCS, vol. 5521, pp. 247–267. Springer, Heidelberg (2009)
9. Baier, C., Klein, J., Klüppelholz, S.: Synthesis of Reo connectors for strategies and controllers. Fundamenta Informaticae **130**(1), 1–20 (2014)
10. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. **61**(2), 75–113 (2006)
11. Becht, M., Gurzki, T., Klarmann, J., Muscholl, M.: ROPE: role oriented programming environment for multiagent systems. In: Cooperative Information Systems (CoopIS 1999), pp. 325–333 (1999)
12. Cabri, G., Ferrari, L., Leonardo, L.: Rethinking agent roles: extending the role definition in the brain framework. In: Systems, Man and Cybernetics (SMC 2004), vol. 6, pp. 5455–5460. IEEE (2004)
13. Cabri, G., Leonardi, L., Ferrari, L., Zambonelli, F.: Role-based software agent interaction models: a survey. Knowl. Eng. Rev. **25**(04), 397–419 (2010)
14. Cabri, G., Leonardi, L., Zambonelli, F.: BRAIN: a framework for flexible role-based interactions in multiagent systems. In: Meersman, R., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 145–161. Springer, Heidelberg (2003)
15. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: synchronisation and context dependency. Sci. Comput. Program. **66**(3), 205–225 (2007)
16. Clarke, D.: A basic logic for reasoning about connector reconfiguration. Fundamenta Informaticae **82**(4), 361–390 (2008)
17. de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Roy, P., Sorea, M.: Sociable interfaces. In: Gramlich, B. (ed.) FroCos 2005. LNCS (LNAI), vol. 3717, pp. 81–105. Springer, Heidelberg (2005)
18. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer, Heidelberg (2001)
19. Dijkstra, E.W.: On the role of scientific thought. In: Selected Writings on Computing: A Personal Perspective, pp. 60–66. Springer, New York (1982). Transcribed 1974
20. Fasli, M.: Social interactions in multi-agent systems: a formal approach. In: Intelligent Agent Technology (IAT 2003), pp. 240–246. IEEE (2003)
21. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Multi Agent Systems (ICMAS 1998), pp. 128–135. IEEE (1998)
22. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
23. Guarino, N., Welty, C.A.: A formal ontology of properties. In: Dieng, R., Corby, O. (eds.) EKAW 2000. LNCS (LNAI), vol. 1937, pp. 97–112. Springer, Heidelberg (2000)
24. Hennicker, R., Klarl, A.: Foundations for ensemble modeling – the Helena approach. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software. LNCS, vol. 8373, pp. 359–381. Springer, Heidelberg (2014)

25. Izadi, M., Movaghar, A., Arbab, F.: Model checking of component connectors. In: 31st Annual International Computer Software and Applications Conference, (COMPSAC), pp. 673–675. IEEE Computer Society (2007)
26. Jongmans, S.-S.T.Q., Arbab, F.: Overview of thirty semantic formalisms for Reo. Sci. Ann. Comput. Sci. **22**(1), 201–251 (2012)
27. Klarman, S., Gutiérrez-Basulto, V.: Two-dimensional description logics of context. In: Description Logics (DL 2011), vol. 745. CEUR Workshop Proceedings (2011)
28. Klein, J.: Compositional synthesis and most general controllers. Ph.D. thesis, Technische Universität Dresden (2013)
29. Klein, J., Baier, C., Klüppelholz, S.: Compositional construction of most general controllers. Acta Informatica, Spec. Issue: Combining Compositionality Concurrency: Part 2 **52**(4–5), 443–482 (2015)
30. Klüppelholz, S.: Verification of branching-time and alternating-time properties for exogenous coordination models. Ph.D. thesis, Technische Universität Dresden (2012)
31. Klüppelholz, S., Baier, C.: Alternating-time stream logic for multi-agent systems. Sci. Comput. Program. **75**(6), 398–425 (2010)
32. Kokash, N., Krause, C., de Vink, E.P.: Data-aware design and verification of service compositions with Reo and mCRL2. In: Symposium on Applied Computing (SAC 2010), pp. 2406–2413. ACM (2010)
33. Kokash, N., Krause, C., de Vink, E.P.: Reo + mCRL2: a framework for model-checking dataflow in service compositions. Formal Aspects Comput. **24**(2), 187–216 (2012)
34. Krause, C., Maraikar, Z., Lazovik, A., Arbab, F.: Modeling dynamic reconfigurations in Reo using high-level replacement systems. Sci. Comput. Program. **76**(1), 23–36 (2011)
35. Kühn, T., Böhme, S., Götz, S., Aßmann, U.: A combined formal model for relational context-dependent roles. In: Software Language Engineering (SLE 2015) (2015, to appear)
36. Kühn, T., Leuthäuser, M., Götz, S., Seidl, C., Aßmann, U.: A metamodel family for role-based modeling and programming languages. In: Combemale, B., Pearce, D.J., Barais, O., Vinju, J.J. (eds.) SLE 2014. LNCS, vol. 8706, pp. 141–160. Springer, Heidelberg (2014)
37. Moon, Y.-J., Silva, A., Krause, C., Arbab, F.: A compositional model to reason about end-to-end QoS in stochastic Reo connectors. Sci. Comput. Program. **80**, 3–24 (2014)
38. Papadopoulos, G.A., Arbab, F.: Coordination models and languages. Adv. Comput. **46**, 329–400 (1998)
39. Pourvatan, B., Sirjani, M., Hojjat, H., Arbab, F.: Symbolic execution of Reo circuits using constraint automata. Sci. Comput. Program. **77**(7–8), 848–869 (2012)
40. Proença, J., Clarke, D.: Data abstraction in coordination constraints. In: Canal, C., Villari, M. (eds.) ESOCC 2013. CCIS, vol. 393, pp. 159–173. Springer, Heidelberg (2013)
41. Reenskaug, T., Wold, P., Lehne, O.A.: Working with Objects: The Ooram Software Engineering Method. Manning, Greenwich (1996)
42. Ren, S., Yu, Y., Chen, N., Marth, K., Poirot, P.-E., Shen, L.: Actors, roles and coordinators — a coordination model for open distributed and embedded systems. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 247–265. Springer, Heidelberg (2006)
43. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng. **35**(1), 83–106 (2000)

44. Talcott, C., Sirjani, M., Ren, S.: Comparing three coordination models: Reo, ARC, and RRD. In: Foundations of Coordination Languages and Software Architectures (FOCLASA 2007). Electronic Notes in Theoretical Computer Science, vol. 194, no. 4, pp. 39–55. Elsevier (2008)
45. Zhu, H., Zhou, M.: Roles in information systems: a survey. IEEE Trans. Syst. Man Cybern. Part C **38**(3), 377–396 (2008)