Thomas Kühn, Kay Bierzynski, Sebastian Richly, Uwe Aßmannn

FRaMED: full-fledge role modeling editor (tool demo)

Erstveröffentlichung in / First published in:

SLE '16: Software Language Engineering. New York, 31.10.–1.11.2016. ACM Digital Library, S. 132–136. ISBN 978-1-4503-4447-0.

DOI: https://doi.org/10.1145/2997364.2997371

Final edited form was published in "SLE '16: Software Language Engineering. Amsterdam 2016", S. 132–136. ISBN: 978-1-4503-4447-0
https://doi.org/10.1145/2997364.2997371

FRaMED: Full-Fledge Role Modeling Editor (Tool Demo)

Thomas Kühn, Kay Bierzynski
Software Technology Group
TU Dresden, Germany
thomas.kuehn3@tu-dresden.de
kay.bierzynski@mailbox.tu-dresden.de

Sebastian Richly, Uwe Aßmannn
Software Technology Group
TU Dresden, Germany
sebastian.richly@rain-it.eu
uwe.assmann@tu-dresden.de

Abstract

Since the year 1977, role modeling has been continuously investigated as promising paradigm to model complex, dynamic systems. However, this research had almost no influence on the design of todays increasingly complex and context-sensitive software systems. The reason for that is twofold. First, most modeling languages focused either on the behavioral, relational or context-dependent nature of roles rather than combining them. Second, there is a lack of tool support for the design, validation, and generation of role-based software systems. In particular, there exists no graphical role modeling editor supporting the three natures as well as the various proposed constraints. To overcome this deficiency, we introduce the *Full-fledged Role Modeling Editor* (FRaMED), a graphical modeling editor embracing all natures of roles and modeling constraints featuring generators for a formal representation and source code of a role-based programming language. To show its applicability for the development of role-based software systems, an example from the banking domain is employed.

Categories and Subject Descriptors I.6.4. [*Simulation and Modeling*]: Model Validation and Analysis—Role-based Modeling; I.6.5. [*Simulation and Modeling*]: Model Development—Formal Modeling

Keywords Role-based Modeling

1. Introduction

Current software systems are characterized by increased complexity, dynamism and context-dependence [18]. Hence, researchers and practitioners demand new concepts beyond object-oriented design able to embrace the dynamic and context-dependent behavior of these systems.

DOI: http://dx.doi.org/10.1145/2997364.2997371

Roles are a natural concept to represent the dynamic, context-dependent behavior of collaborating objects [20]. Since its introduction in 1977 [1], role-based modeling has been continuously investigated as promising paradigm to model complex, dynamic, and context-dependent systems [13, 20, 23]. Accordingly, multiple *role-based modeling languages* (RML) have been proposed in the past, ranging from data modeling [1, 8, 12, 16] via conceptual modeling [7, 23] through to software architecture modeling [9, 19]. This research, however, had only marginal impact on the design of todays complex and context-adaptive software systems.

To enable the use of RMLs for conceptual modeling to both researchers and practitioners, two major blocking factors have to be addressed. First, most RMLs focus either on the behavioral, relational or context-dependent nature of roles [14] rather than combining them into one coherent model. Second, there is a lack of tool support for the design, validation, and generation of role-based software systems. In particular, there exists no graphical editor for role-based modeling supporting all natures of roles. Although some approaches permit using a graphical editor, e.g. [7–9, 24], none of these editors encompasses all natures of roles [14]. To overcome the former deficiency, the *Compartment Role Object Model* (CROM) was introduced as formal model combining all natures as well as various modeling constraints [14]. To address the second issue accordingly, we introduce the first *Full-fledged Role Modeling Editor* (`FRaMED`), a graphical modeling editor for CROMs embracing all natures and proposed modeling constraints featuring distinct code generators generating either a formal representation of CROM [14] or source code of the *Scala Roles Language* (`SCROLL`) [15]. To show its applicability for conceptual modeling, we employed an example from the banking domain. In conclusion, `FRaMED` provides all means necessary to allow both researchers and practitioners to model, reason about, and implement role-based software systems.[1]

The paper is structured as follows: Sect. 2 presents the natures of roles found in role-based languages. Afterwards,

[1] The approved artefact of FRaMED is available via: `http://st.inf.tu-dresden.de/intern/framed/framed-ubuntu.ova`

Final edited form was published in "SLE '16: Software Language Engineering. Amsterdam 2016", S. 132–136. ISBN: 978-1-4503-4447-0
https://doi.org/10.1145/2997364.2997371

Sect. 3 briefly discusses related graphical RMLs. Sect. 4 describes FRaMED's architecture and employed frameworks. Sect. 5 showcases the design of a role-based conceptual model for a banking application and finally, Sect. 6 concludes the paper.

2. Nature of Roles

Roles are not a new concept. Yet, there is still no common understanding of roles in the literature [13, 23]. On the contrary, [23] and [13] identified 26 features attributed to roles that we group into the following three natures.

The **behavioral nature** establishes that unrelated objects can play roles and roles adapt the behavior of playing objects [13, 23]. Additionally, objects can play roles of a different type multiple times. Consider, for instance, the role of a *customer* of a bank that can be played by either a *person* or a *company* and allows them to make transactions, deposits, and withdrawals. Obviously, one *person* can be a *customer* in several banks. This nature is usually captured by the *fills*-relation between classes and role types denoting those classes whose objects can play roles of the given type. In contrast, the **relational nature** states that roles denote the binding ends of relationships. This nature is present in most modeling languages, e.g. ER [4] and UML [22]. Still, these languages do not foster the dynamism and flexibility of roles, as roles degenerate to named placeholders. Hence, [1, 2, 8, 11, 23] introduced roles tied to relationships as first-class citizens permitting them to be played by unrelated objects and having relationship specific properties. To model that *consultants advise customers*, one can specify a relationship type *advises* between the *consultant* and *customer* role types and add relationship specific fields. However, all these modeling languages assume that relationships are context-independent and cannot play roles themselves. Thus, *transactions* between *accounts* managed by a *bank* cannot be modeled properly. To resolve this, RMLs have incorporated the **context-dependent** nature of roles, e.g. [6, 9, 12, 20], that characterizes roles and relationships as context-dependent. Both are encapsulated in a *context* as definitional boundary. Yet, different approaches use different terms for this conceptual entity. Consequently, *compartments* were introduced in [13] to generalize the different terms. Compartments can have properties and behavior, as well as play roles themselves. In accordance, a *transaction* for the transferal of money from a *source* to a *target* account would be modeled as compartment type and would play the role of a *money transfer* within a *bank* compartment. In contrast to context-dependent roles, only few approaches also include context-dependent relationships, e.g. [9, 12].

Including these natures into one RML already leads to a rich modeling framework, yet its expressiveness is determined by the available kinds of **constraints**. These range from classical *relationship cardinalities*, e.g. [4, 22], over mathematical *relationship constraints*, e.g. [2, 8], to *role constraints*, e.g. [9, 21, 25]. Within a *bank* compartment type, for instance, one could specify that each *consultant* advises at least one *customer* with relationship cardinalities, that no *consultant* advises himself as a *customer* with an intra-relationship constraint, and that each *bank* must have at least one *consultant* [14]. Although various RMLs have introduced different kinds of constraints, only CROM [14] includes most of them. Consequently, FRaMED is founded on its notation and definitions.

Role-Based Languages	Behavioral	Relational	Contextual	Graphical	Editor Support
Lodwick (2000) [23]	●	●	○	●	○
Generic Role Model (2002) [5]	●	○	○	◐	○
ORM 2 (2005) [8]	●	●	○	●	●
E-CARGO Model (2006) [25]	◐	○	●	○	○
Metamodel for Roles [6]	●	○	●	○	○
INM (2009) [16]	●	◐	●	●	○
DCI (209) [20]	●	○	●	○	○
Onto-UML (2012) [7]	◐	◐	◐	●	⊞
Helena Approach (2014) [9]	●	●	●	●	⊞
RSQL (2016) [12]	●	●	●	●	○
SCROLL (2015) [15]	●	○	●	○	○
formal CROM (2015) [14]	●	●	●	●	○
FRaMED	●	●	●	●	●

●: fulfilled, ◐:partly, ○: unfulfilled, ⊞: possible

Table 1. Comparison of RMLs, based on [13].

3. State of the Art

There exist various role-based modeling and programming languages in the literature. Hence, we refer the reader to [23], [26] and [13] for detailed surveys on role-based languages. For brevity, Tab. 1 compares the related RMLs with FRaMED by means of their supported nature of roles, their graphical notation, and editor support. Henceforth, we focus on those RMLs that provide either a graphical editor or rely on standardized graphical notations. Up to our best knowledge, *Object-Role Modeling* (ORM) 2 [8] is the only modeling language that provides a graphical editor. However, it only supports the relational nature of roles. In contrast, the following approaches only introduce a graphical notation for roles. First, the *Information Networking Model* (INM) [16] focuses on the context-dependent nature of roles introducing context and roles as model elements as well as multiple kinds of relations among them. In contrast to FRaMED, their notation is very cluttered mixing entities, context-dependent

Final edited form was published in "SLE '16: Software Language Engineering. Amsterdam 2016", S. 132–136. ISBN: 978-1-4503-4447-0
https://doi.org/10.1145/2997364.2997371

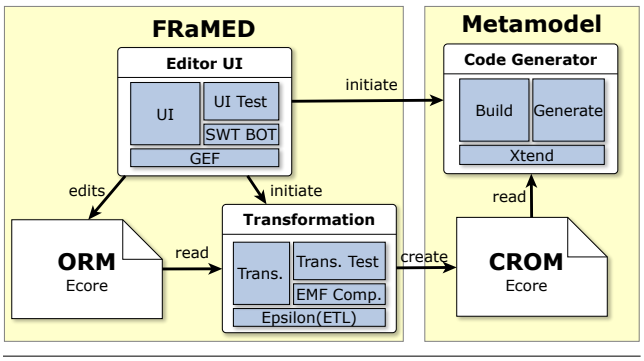

Figure 1. Architecture

roles, and relationships. Second, *Onto-UML* [7] is an ontologically founded modeling language introducing role types and relationships. It employs stereotypes to extend classical UML class diagrams. Last but not least, the *Helena Approach* [9] is an architecture modeling language incorporating both the relational and the context-dependent nature of roles by providing *ensembles* as context for roles. Similarly, its notation relies on stereotypes for the basic model elements and a small extension for occurrence constraints. While both are usable with any UML editor, they are hard to comprehend, as they require reading the stereotype annotations [17]. In turn, FRaMED's graphical notation is greatly inspired by [3, 10, 21, 22] and the guidelines in [17].

4. Architecture

Generally, FRaMED is built on the Eclipse platform[2] and is available on *GitHub*.[3] It follows a model-driven approach and employs the *Eclipse Modeling Framework* (EMF).[4] Fig. 1 provides an overview of its software architecture.

For its development, we picked a suitable configuration of the *family of metamodels for RMLs* [13] and generated the corresponding *Ecore* metamodel within a separate plugin.[5] As such, this metamodel only captures the structure of CROMs and is void of any layout information. To decouple FRaMED from this CROM metamodel, the editor, again, is implemented as plugin that only emits instances of the CROM metamodel. The *Editor UI* handles all user interactions and is implemented employing the *Graphical Editing Framework* (GEF).[6] Internally, FRaMED uses a custom *Ecore* metamodel for the graphical representation of CROM, denoted *Object Relation Model* (ORM). This metamodel is tailored towards the graphical aspects of CROM, such as *shapes*, *relations*, *segments*, and *bend points*. On saving a ORM instance (*.crom_dia file), another plugin is tasked with its transformation to the corresponding CROM instance (*.crom file). The *Transformation* plugin, in turn, utilizes Epsilon,[7] a rule-based model-to-model transformation engine, to declaratively specify the translation of ORM instances to CROM instances. Once a CROM instance is created, a user can trigger the *Code Generator* plugin to either generate a formal CROM, based on the reference implementation in [14], or generate SCROLL source code [15]. While the former can be used to validate a CROM instance, the latter can be completed to a working role-based application. Last but not least, we employed JUnit 4, EMF Compare, and SWTBot to setup a test infrastructure for FRaMED.

Although this architecture is rather complex, it facilitates separation of concerns by establishing the CROM metamodel as central representation of the RML. This not only permits the separate evolution of the metamodel, editor, and code generators, but also the development of further tools, such as a text-based editor for CROM or a code generator for Object Teams [10].

5. Use Case

To explain the role modeling workflow, we designed a small banking application, extracted from [20], as our running example. In this scenario, a *bank* is a financial institution that employs *consultants*, serves *customers*, and handles *money transfers*. *Consultants* can be *persons* whereas the latter either *companies* or *persons*. Additionally, a consultant *advises* at least one customer and customers can *own* multiple *savings* and *checking accounts*. Moreover, customers can initiate transactions to transfer money from one source account to another target account.

To design a conceptual model of the banking application, each of the above domain concepts must be sorted into either *natural types*, *role types*, *data types*, *compartment types*, or *relationship types*. Following the ontological foundation in [14], Bank and Transaction are modeled as compartment types, as well as Person, Companies and Accounts as natural types. In the top-level view of FRaMED one can create natural, data, and compartment types; specify their inheritance relation; as well as create and refine the fills-relation. After adding these model elements by dragging them to the canvas and naming them accordingly, we *"step into"* each compartment type to model its internals. Within a compartment type one can create role types, role groups, specify various role constraints, create relationship types between two role types, and specify inter-relationship constraints, add intra-relationship constraints. Accordingly, we add the Consultant, Customer, SavingsAccount, and CheckingAccount as role types, as well as advises, own_sa and own_ca as relationship types to the Bank. Similarly, a Source role type, a Target role type, and the trans relationship type are added to the Transaction, highlighted in Fig. 2. Last but not least, we "step out" to the top-level

[2] https://eclipse.org/

[3] https://github.com/leondart/FRaMED/releases/tag/v2.0.3

[4] https://eclipse.org/modeling/emf/

[5] https://github.com/Eden-06/CROM

[6] https://eclipse.org/gef/

[7] http://www.eclipse.org/epsilon/

Final edited form was published in "SLE '16: Software Language Engineering. Amsterdam 2016", S. 132–136. ISBN: 978-1-4503-4447-0
https://doi.org/10.1145/2997364.2997371

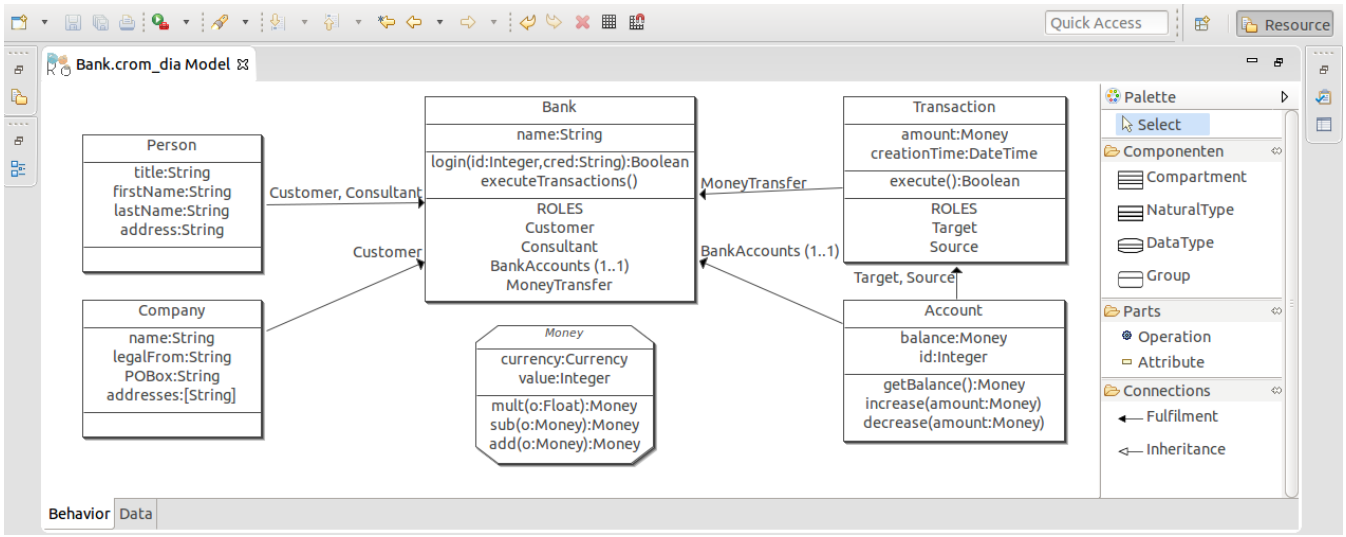

Figure 2. Banking example in FRaMED.

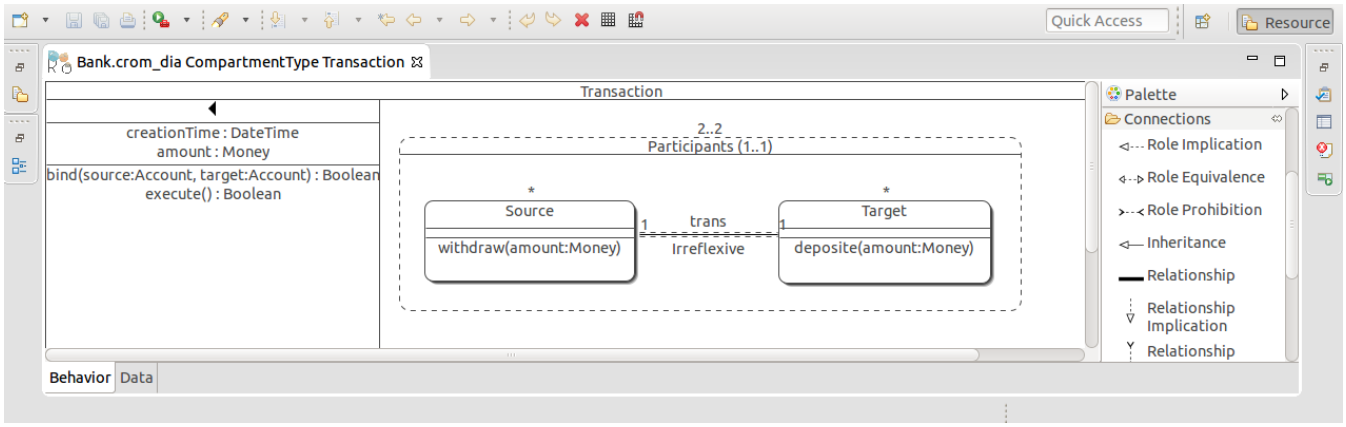

Figure 3. Focus view of the Transaction compartment type.

view to link the natural and compartment types to those role types they can play. First, the *Fulfillment* relation must be selected then the relation can be drawn from a natural and compartment types to those compartment types containing the role types to be filled. Finally, a right click on the individual fulfillment relation opens up the "Fulfill Roles" dialog, where the playable role types can be selected. Following these steps, we finally come to the conceptual model for the banking application, depicted in Fig. 3, capturing not only the dynamics of the banking domain, but also its constraints.

Whenever the role model is saved, the corresponding `crom` file is generated. Upon right-clicking on this file, the *"Generate"* item is available in the context menu expanding to *"SCROLL Code"* and *"Formal CROM"*. Each triggers the respective code generator. Both generated files could then be executed with the corresponding runtime library. In sum, the presented workflow can be easily reproduced and tailored to more elaborate use cases.

6. Conclusion

This paper presented FRaMED, a graphical role modeling editor for CROM [14]. Thus, it is the first modeling editor for role-based domain models that supports all natures of roles and constraints presented in the literature. It enables the design of role-based software systems by providing additional means for validation and code generation. It is open source and freely available to let both researchers and practitioners harness the power of role-based modeling. In the future, we will extend FRaMED, to not only support one member of the *metamodel family for RMLs* [13], but all of them.

Acknowledgments

This work is funded by the German Research Foundation (DFG) within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907). Special thanks go to Lars Schütze, Paul Peschel, David Gollasch, Johannes Tandler, and Duc Dung Dam for their contributions.

Final edited form was published in "SLE '16: Software Language Engineering. Amsterdam 2016", S. 132–136. ISBN: 978-1-4503-4447-0
https://doi.org/10.1145/2997364.2997371

References

[1] C. W. Bachman, M. Daya, C. W. Bachman, and M. Daya. The role concept in data models. In *Proceedings of the third international conference on Very Large Data Bases*, volume 3, pages 464–476, 1977.

[2] S. Balzer and T. R. Gross. Verifying multi-object invariants with relationships. In *ECOOP 2011–Object-Oriented Programming*, pages 358–382. Springer, 2011.

[3] S. Balzer, T. Gross, and P. Eugster. A relational model of object collaborations and its use in reasoning about relationships. In E. Ernst, editor, *ECOOP*, volume 4609 of *Lecture Notes in Computer Science*, pages 323–346. Springer, 2007. ISBN 978-3-540-73588-5.

[4] P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[5] M. Dahchour, A. Pirotte, and E. Zimányi. A generic role model for dynamic objects. In *Advanced Information Systems Engineering*, pages 643–658. Springer, 2002.

[6] V. Genovese. A meta-model for roles: Introducing sessions. In *Proceedings of the 2nd Workshop on Roles and Relationships in Object Oriented Programming, Multiagent Systems, and Ontologies*, pages 27–38. Technische Universität Berlin, 2007.

[7] G. Guizzardi and G. Wagner. Conceptual simulation modeling with onto-uml. In *Proceedings of the Winter Simulation Conference*, page 5. Winter Simulation Conference, 2012.

[8] T. Halpin. ORM 2. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 676–687. Springer, 2005.

[9] R. Hennicker and A. Klarl. Foundations for ensemble modeling–the helena approach. In *Specification, Algebra, and Software*, pages 359–381. Springer, 2014.

[10] S. Herrmann. Programming with roles in ObjectTeams/Java. *AAAI Fall Symposium Roles, an interdisciplinary perspective*, 2005.

[11] T. Jäkel, T. Kühn, S. Hinkel, H. Voigt, and W. Lehner. Relationships for dynamic data types in RSQL. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, 2015.

[12] T. Jäkel, T. Kühn, H. Voigt, and W. Lehner. Towards a contextual database. In *20th East-European Conference on Advances in Databases and Information Systems*, 2016.

[13] T. Kühn, M. Leuthäuser, S. Götz, C. Seidl, and U. Aßmann. A metamodel family for role-based modeling and programming languages. In *Software Language Engineering*, volume 8706 of *Lecture Notes in Computer Science*, pages 141–160. Springer, 2014.

[14] T. Kühn, S. Böhme, S. Götz, and U. Aßmann. A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 113–124. ACM, 2015.

[15] M. Leuthäuser and U. Aßmann. Enabling view-based programming with scroll: Using roles and dynamic dispatch for etablishing view-based programming. In *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, pages 25–33. ACM, 2015.

[16] M. Liu and J. Hu. Information networking model. In *Conceptual Modeling-ER 2009*, pages 131–144. Springer, 2009.

[17] D. Moody. The physics of notations: toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, 35(6):756–779, 2009.

[18] S. Murer, C. Worms, and F. J. Furrer. Managed evolution. *Informatik-Spektrum*, 31(6):537–547, 2008.

[19] C. Piechnick, S. Richly, S. Götz, C. Wilke, and U. Aßmann. Using role-based composition to support unanticipated, dynamic adaptation-smart application grids. In *ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications*, pages 93–102, 2012.

[20] T. Reenskaug and J. O. Coplien. The dci architecture: A new vision of object-oriented programming. *An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html*, 2009.

[21] D. Riehle and T. Gross. Role model based framework design and integration. In *Proceedings OOPSLA '98, ACM SIGPLAN Notices*, pages 117–133, Oct. 1998.

[22] J. Rumbaugh, R. Jacobson, and G. Booch. *The Unified Modelling Language Reference Manual*. Addison-Wesley, 1st edition, 1999.

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

[24] F. Steimann. A radical revision of UML's role concept. In *UML 2000 - The Unified Modeling Language*, pages 194–209. Springer, 2000.

[25] H. Zhu and M. Zhou. Role-based collaboration and its kernel mechanisms. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(4):578–589, 2006.

[26] H. Zhu and M. Zhou. Roles in information systems: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(3):377–396, 2008.