Spezifikation von Softwareautonomie

SPEZIFIKATION VON SOFTWAREAUTONOMIE

Gerhard Weiß, Johann Duscher, Matthias Nickles, Michael Rovatsos

Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching
{weissg,duscher,nickles,rovatsos}@in.tum.de

ZUSAMMENFASSUNG

Gegenstand des agentenorientierten Software Engineering ist die Entwicklung von Software, deren Struktur und Funktionalität aus dem Blickwinkel einer Menge von Agenten betrachtet wird. Unter einem Agent wird dabei eine autonome, flexible und kooperative Softwareeinheit verstanden. Eine zentrale Herausforderung des agentenorientierten Software Engineering ist die genaue Erfassung der Art und des Umfangs der Autonomie, den ein agentenorientiertes Softwaresystem beziehungsweise ein Softwareagent besitzen soll. Dieser Beitrag stellt eine Sprache (**RNS+**) und ein dazugehöriges Tool (**CRNS**) zur Autonomiespezifikation vor. **RNS+** zeichnet sich durch hohe Präzision und Ausdrucksstärke aus; **CRNS** unterstützt die Erstellung von **RNS+**-basierten Spezifikationen und automatisiert die Erkennung und Beseitigung von Konflikten, die zwischen Softwareagenten aufgrund ihrer Autonomie vorliegen können.

1 EINLEITUNG

Agentenorientiertes Software Engineering ist ein Gebiet, das an der Schnittstelle zwischen Software Engineering einerseits und Agenten- und Multiagententechnologie andererseits angesiedelt ist und seit einigen Jahren ein rasch wachsendes Interesse erfährt (z.B. [6,9,16,18]). Gegenstand dieses Gebietes sind Implementierungssprachen, Vorgehensweisen, Techniken und Tools für die Entwicklung, die Analyse und den Einsatz von agentenorientierter Software, also von Software, deren Struktur und Funktionalität unter dem Blickwinkel einer Menge von Agenten betrachtet wird. Unter einem Agent wird dabei eine abgrenzbare Softwareeinheit verstanden, die autonom, flexibel und in Koordination mit anderen Agenten Ziele verfolgt, die vom Systementwickler oder Systembenutzer vorgegeben sind. Agentenorientierung eignet sich in besonderem Maß für die Realisierung einer wichtigen Klasse von komplexen Softwareanwendungen, nämlich solchen, die sich durch Verteiltheit, Offenheit und Einbettung in dynamische soziotechnische Umgebungen auszeichnen. Bereiche, für die dieser Anwendungstyp charakteristisch ist, sind beispielsweise E/M-commerce, E/M-business, Ubiquitous Computing und Wissens- und Informationsmanagement – generell also Bereiche, in denen die zunehmende Rechnervernetzung und Plattforminteroperabilität von großer Bedeutung ist für die Realisierung neuer und die Erweiterung bestehender Informations- und Geschäftsprozesse. Agentenorientierung zeichnet sich, vor allem auch im Vergleich zu den traditionellen Paradigmen der Struktur- und Objektorientierung, durch ihren Fokus auf Autonomie als gewollte Systemeigenschaft aus. Der Kerngedanke, der diesem Fokus zugrundeliegt, ist es, Anwendungssysteme mit der Fähigkeit zu autonomen Verhalten auszustatten und sie dadurch in die Lage zu versetzen, ihre Komplexität möglichst selbständig handzuhaben. Die Relevanz, die diesem Kerngedanken zukommt, zeigt sich auch daran, dass er – in unterschiedlicher Intensität und Ausrichtung („self-governing", „self-structuring", „self-healing", „self-reparing" usw.) – seit Kurzem von führenden Vertretern der IT-Branche in verschiedenen Initiativen propagiert wird. Zu nennen ist hier insbesondere IBM's autonomic computing Initiative [8], aber auch Sun's N1 Initiative [15], HP's adaptive enterprise Initiative [7] und Microsoft's dynamic systems Initiative [12].

Zu den entscheidenden Aufgaben bei der Entwicklung eines agentenorientiertes Softwaresystems gehört die Spezifikation der Art und des Umfangs der Autonomie, welche die im System involvierten Softwareagenten besitzen sollen. Dieser Beitrag stellt eine Sprache, **RNS+** („Roles, Norms, Sanctions"), und ein zugehöriges Tool, **CRNS** („conflicts + RNS"), für Autonomiespezifikation vor. Die der Sprache **RNS+** zugrundeliegende Sicht ist, dass Softwareagenten als Inhaber von Rollen agieren um ihre Ziele zu erreichen; als Rolleninhaber sind die Agenten bestimmten Verhaltensnormen (Erlaubnis, Verpflichtung, Verbot) ausgesetzt, wobei Normkonformität und –abweichung sanktioniert (belohnt bzw. bestraft) werden können. **RNS+** unterscheidet sich von vergleichbaren Spezifikationsansätzen durch eine sehr hohe Präzision und Ausdruckstärke. **CRNS** erlaubt eine komfortable **RNS+**-basierte Autonomiespezifikation von Softwareagenten. Insbesondere automatisiert **CRNS** die Handhabung – Identifikation und Auflösung – von normbasierten Konflikten (z.B. gleichzeitige Verpflichtung und Verbot zur Durchführung einer Aktivität) zwischen Softwareagenten.

Dieser Beitrag ist wie folgt strukturiert. Abschnitt 2 stellt die Sprache **RNS+** vor. Die Handhabung von Konflikten wird in Abschnitt 3 behandelt. Abschnitt 4 beschreibt das Tool CRNS. Abschnitt 5 fasst die wesentlichen Merkmale von **RNS+** und **CRNS** zusammen und verweist auf verwandte Arbeiten. Abschnitt 6 schließt den Beitrag mit allgemeineren Anmerkungen zur Autonomiespezifikation.

2 RNS+ – Eine Sprache Zur Spezifikation Von Autonomie

Der Sprache **RNS+** liegt folgende Systemsicht zugrunde. Die in einem Softwaresystem wirkenden Softwareagenten sind in eine soziale Umgebung eingebettet, die deren Verhaltensräume regulieren beziehungsweise begrenzen soll. Diese Umgebung setzt sich aus einer Menge von Rollen (Rollenraum), zusammen, welche die Agenten annehmen müssen, um ihre individuellen oder gemeinsamen Ziele zu erreichen. Jeder Agent kann jederzeit eine beliebige Menge von Rollen gleichzeitig einnehmen, wobei jeder Rolle mindestens eine Aktivität zugeordnet ist. Zu jeder Aktivität gehören Normen (Rechte, Pflichten, Verbote) und Sanktionen (Belohnungen, Bestrafungen). Während Normen das zu erwartende Verhalten eines Rolleninhabers beschreiben, sind durch die Sanktionen Konsequenzen für normkonformes und -abweichendes Verhalten festgelegt. Sanktionen drücken gleichsam das Maß an Kontrolle aus, das auf Agenten ausgeübt werden kann. Wichtig ist, dass mit dem Rollenkonzept nicht das tatsächliche, sondern das *gewünschte* Verhalten spezifiziert wird. Das verwendete Rollenkonzept macht damit das zukünftige Verhalten von Softwareagenten vorhersagbarer und kalkulierbarer, determiniert es jedoch nicht völlig und schließt insbesondere die Möglichkeit von normabweichenden Verhalten nicht aus (was ja auch dem Konzept der Verhaltensautonomie widersprechen würde).

Die Sprache **RNS+**, die in der Verwendung ihrer Grundkonzepte (Rolle, Norm, Sanktion) an die soziologische Rollentheorie [3] angelehnt ist, stellt eine Weiterentwicklung des Spezifikationsschemas **RNS** [13,17] dar. Im Gegensatz zu **RNS** eignet sich **RNS+** zur automatisierten Konflikthandhabung; für diesen Zweck wurde **RNS** entsprechend erweitert und formal präzisiert. Im Folgenden wird **RNS+** im Detail beschrieben. Zur Illustration der wesentlichen Sprachelemente wird dabei das Anwendungsszenario „*Elektronische Verkaufsplattform*" verwendet, wobei auf dieser Plattform Softwareagenten für ihre menschlichen Benutzer Käufe und Verkäufe von Waren tätigen.

2.1 Formale Konzepte und Konstrukte

Dieser Abschnitt beschreibt die formalen Aspekte von **RNS+** und die Syntax dieser Sprache in einer BNF-ähnlichen Notation. In der verwendeten Notation sind alle zur Spezifikationssprache gehörenden Terminale mit **`diesem`** Schriftstil, und Nichtterminale mit *`diesem`* Schriftstil kenntlich gemacht. Zur Notationssprache selbst gehörende Zeichen sind durch `diesen` Schriftstil gekennzeichnet. Optionale Elemente sind mit den Zeichen `[` und `]` geklammert. Alternative Elemente sind durch das Zeichen `|` voneinander getrennt und ggf. mit den Zeichen `(` und `)` geklammert.

Soziales Umfeld („role space")

In **RNS+** wird ein soziales Umfeld als Menge von Rollen (Rollenraum) spezifiziert:
role-space-spec ::= **role_space** *role-space-id* **{** *role-id-list* **}**
role-space-id ::= eindeutiger Rollenraum-Bezeichner

Beispiel 1 Definition eines sozialen Umfeldes

Rollen sind vorgesehen für Besucher der elektronischen Verkaufsplattform, potentielle Käufer und Verkäufer, sowie für Katalog-Manager und Plattform-Manager.

```
role_space selling_platform {
    visitor, buyer, seller, catalog_mg, platform_mg
}
```

Soziale Rolle („role")

Eine Rolle ist einem sozialen Umfeld zugeordnet, und ihre Spezifikation enthält eine nicht leere Menge von Aktivitäten:
role-spec ::= **role** *role-id* **{** *activity-spec-list* **}**
role-id ::= eindeutiger Rollen-Bezeichner

Beispiel 2 Definition einer sozialen Rolle

Die Rolle eines Käufers.

```
role buyer {
    act request propose_selling_contract(…) {…};
    act basic pay_for_article(…) {…};
}
```

Variablen

Variablen treten innerhalb einer **RNS+** Spezifikation in zwei verschiedenen Bereichen auf:

1. innerhalb des Bedingungsteils von Norm-Spezifikationen und
2. im Header derjenigen Aktivitätsspezifikationen, welche Variablenlisten.

In **RNS+** kann einer Variablen ein genau definierter Wertebereich zugeordnet werden:
variable-spec ::= *variable-id*[**{***domain-elem-list***}**]
domain-elem ::= *symbol-value* | *number-value* | *number-value***..***number-value*

Beispiel 3 Definition einer Variable

Eine Variablenspezifikation `MixedVar` mit numerischen und symbolischen Werten in ihrem Wertebereich.

```
MixedVar{3, 10, 1976, 2002..2007, duscher, nickles, rovatos, weissg}
```

Norm-Sanktions-Paare („status statements")

Einerseits ermöglichen Norm-Sanktions-Paare die explizite Angabe von aktivitätsspezifischen Normen und Sanktionen, die für einen Inhaber einer Rolle A gelten sollen, wenn er durch einen Inhaber einer Rolle B aufgefordert wird, eine zur Rolle A gehörende Aktivität auszuführen bzw. zu unterlassen. Andererseits kann auch spezifiziert werden, welchen aktivitätsspezifischen Normen und Sanktionen ein Inhaber der Rolle A *unabhängig* von Handlungsaufforderungen durch andere Rolleninhaber „ausgesetzt" ist. Diese „unabhängigen" Normen und Sanktionen zielen also auf die Regulierung dessen ab, was ein Rolleninhaber von sich aus (ohne aufgefordert zu werden) machen darf/machen muss/zu unterlassen hat:
status-statement ::= **<***status-type***>** **:** **<***norm-spec***>** **+** **<***sanction-spec***>**

Der Status-Typ

Ein status statement ist immer entweder abhängig („**dep**endent") oder unabhängig („**ind**ependent") von Handlungsaufforderungen anderer Rolleninhaber:

status-type ::= **<ind>** | **<dep** (*role-id* | **EACH**)**>**

wobei das Schlüsselwort **EACH** für eine beliebige Rolle steht.

Die Normspezifikation

Eine Normspezifikation assoziiert ein Recht („**p**ermission"), eine Verpflichtung („**o**bligation"), oder ein Verbot („**i**nterdiction") mit einer Bedingung:

norm-spec ::= **norm** **<**(**p** | **o** | **i**)**>** **<***boolean-expression***>**

Das Nichtterminal *boolean-expression* steht für einen booleschen Ausdruck, der sich i.d.R. aus Variablen und entsprechenden Vergleichsoperatoren zusammensetzt. Erst wenn sich dieser Ausdruck mittels einer passenden Variablenbelegung mit *„wahr*" interpretieren lässt, ist die Normspezifikation für weitere Betrachtungen (z.B. Konfliktanalyse) *relevant* bzw. *aktiv*.

Die Sanktionsspezifikation

Eine Sanktionsspezifikation kann zum einen entweder eine Belohnung („**re**ward"), oder eine Bestrafung („**pu**nishment") repräsentieren. Zum anderen kann über sie eine Sanktion auch explizit ausgeschlossen werden („**no**ne"):

sanction-spec ::= **sanc** **<**(**no** | **re** | **pu**)**>** **<**(**no** | *sanction-id*)**>**

Beispiel 4 Definition eines status statement

Das Missachten einer Unterlassungsaufforderung durch einen `platform_mg` wird mit dem Entzug der Rolle bestraft.

```
<dep platform_mg> : norm <i> <true> + sanc <pu> <lose_role>
```

Aktivitäten

Jeder Rolle ist eine nicht-leere Menge von Aktivitäten zugeordnet, die ein Rolleninhaber ausführen kann. Jede dieser Aktivitäten besitzt eine sog. status range. Die *status range* ist die Menge aller status statements, die die jeweilige Aktivität betreffen:

status-range-spec ::= **status_range** **{** *status-statement-list* **}**

RNS+ unterscheidet vier verschiedene Aktivitätstypen, welche in den folgenden Abschnitten erläutert werden.

Basisaktivitäten (Typ I)

Basisaktivitäten sind elementare, anwendungs- und domänenspezifische Aktivitäten, die der Handhabung von Ressourcen und Ereignissen dienen:

basic-activity-spec ::= **act** **basic** *basic-activity-id***(***variable-list***)** **{**
status-range-spec
}

Beispiel 5 Definition einer Basisaktivität

Der Verkäufer `seller` akzeptiert mit dem Ausführen der Aktivität `accept_selling_contract` einen Kaufantrag (durch einen `buyer` gestellt). Die Ausführung der Aktivität ist erlaubt, wenn der Käufer mind. so alt ist, wie das für den Artikel festgesetzte Mindestalter. Ein Verstoß bedeutet den Verlust seiner Rolle.
Auf Antrag des Käufers ist der Verkauf des Artikels erlaubt, falls das Gebot des Käufers unterhalb des ausgeschriebenen Preises liegt. Verpflichtend ist der Verkauf des Artikels an den Käufer jedoch, wenn das Gebot mind. dem Betrag des ausgeschriebenen Preises entspricht.

```
act basic accept_selling_contract(
    Buyer_Id, Buyer_Age{16..150}, Article_Id, Article_FSK{6..21},
    Article_Price, BitPrice)
{
    status_range {
        <ind> : norm <p> <Buyer_Age >= Article_FSK>
              + sanc <pu> <lose_role>;
        <dep buyer> : norm <p> <BitPrice < Article_Price>
                    + sanc <re> <get_laudation>;
        <dep buyer> : norm <o> <BitPrice >= Article_Price>
                    + sanc <pu> <get_complaint>;
    }
}
```

Handlungsaufforderungen (Typ II)

Dieser Aktivitätstyp erlaubt die explizite Modellierung von Forderungen an andere Agenten, eine bestimmte Aktivität durchzuführen („request") oder zu unterlassen („**not**-request"); i.d.R. werden damit auf der Seite des Empfängers gleichzeitig Normen und Sanktionen induziert. Bzgl. des Typs der geforderten Aktivität existieren keine Einschränkungen. So können insbesondere formal auch Aufforderungsketten („request for request for ...") formuliert werden:

request-activity-spec ::=
    act request *request-activity-id***(**
        *variable-list***;**
        (*agent-id-list* | **EACH**)**;**
        (*role-id-list* | **EACH**)**;**
        [**not**] *activity-call***)**
    {
        status-range-spec
        normative-impact-spec
    }
normative-impact-spec ::= **normative_impact { ***norm-spec-list* **}**

Beispiel 6 Definition einer Handlungsaufforderung

Eine Aktivität der Rolle buyer, mit der ein Käufer einen Kaufantrag für einen Artikel stellen kann.

```
act request propose_selling_contract(
    Buyer_Id, Buyer_Age{16..150},
    Article_Id, Article_FSK{6..21}, Article_Price, BitPrice;
    EACH;
    seller;
    accept_selling_contract(Buyer_Id, Buyer_Age,
                            Article_Id, Article_FSK, Article_Price,
                            BitPrice))
{
    status_range {
        <ind> : norm <p> <true>
              + sanc <no> <no>;
    }
    normative_impact {
        norm <p> <BitPrice < Article_Price>;
        norm <o> <BitPrice >= Article_Price>;
    }
}
```

Sanktionsaktivitäten (Typ III)

Dieser Aktivitätstyp dient der Belohnung oder Bestrafung eines Agenten aufgrund normkonformen beziehungsweise -abweichenden Verhaltens. Insbesondere kann festgelegt werden, welche Rollen überhaupt mit Sanktionsberechtigungen ausgestattet sind:

```
sanction-activity-spec ::=
    act sanction sanction-activity-id(
        (agent-id-list | EACH);
        (role-id-list | EACH);
        (activity-id | EACH);
        (status-statement | EACH))
    {
        status-range-spec
        [sanctioning-impact-spec]
    }
sanctioning-impact-spec ::= sanctioning_impact { sanction-spec-list }
```

Beispiel 7 Definition einer Sanktionsaktivität

Eine Aktivität der Rolle platform_mg, mit der jedes normkonforme bzw. -abweichende Verhalten aller Rolleninhaber sanktionieren werden kann (dazu sind aufgrund der Ausdruckstärke von **RNS+** nur wenige Zeilen notwendig).

```
act sanction sanction_activity(
    EACH; EACH; EACH; EACH)
{
    status_range {
        <ind> : norm <p> <true> + sanc <no> <no>;
    }
}
```

Änderungsaktivitäten (Typ IV)

Mit Hilfe von Änderungsaktivitäten können zu beliebigen Aktivitäten gehörende Normen und Sanktionen während der Laufzeit des Systems geändert werden. Dazu werden neue Norm-Sanktions-Paare zum status range der betreffenden Aktivitäten hinzugefügt bzw. vorhandene werden ersetzt oder entfernt:

```
change-activity-spec ::=
    act change change-activity-id(
        (role-id-list | EACH);
        (activity-id-list | EACH);
        change-activity-cmd)
    {
        status-range-spec
        [status-impact-spec]
    }
change-activity-cmd ::=
    add status-statement
  | delete status-statement
  | replace status-statement by status-statement
status-impact-spec ::= status_impact { change-activity-cmd }
```

Beispiel 8 Definition einer Änderungsaktivität

Eine Aktivität der Rolle `platform_mg`, mit der ein `catalog_mg` dazu verpflichtet werden kann, Artikel hinsichtlich ihrer Legalität zu überprüfen. Dazu wird die Erlaubnis des `catalog_mg` durch eine Verpflichtung ersetzt.

```
act change obligate_legality_check(
    catalog_mg;
    check_for_legality;
    replace <ind> : norm <p> <true> + sanc <no> <no>
         by <ind> : norm <o> <true> + sanc <pu> <get_warning>)
{
    status_range {
        <ind> : norm <p> <true> + sanc <no> <no>;
    }
}
```

3 Handhabung von Konflikten

In diesem Abschnitt wird beschrieben, welche generischen Klassen von Konflikten zwischen Softwareagenten von **RNS+** erfasst und automatisch identifiziert werden können. Weiterhin werden drei anwendungs- und domänenunabhängige Strategien zur Konfliktauflösung vorgestellt.

3.1 Klassifikation von Konflikten

Normbasierte Konflikte

Von zentraler Bedeutung ist die Klasse der normbasierten Konflikte (*Konflikte 1. Ordnung*), da sie praktisch in jeder Anwendung präsent sind. Ein normbasierter Konflikt *kann* immer dann vorliegen, wenn sich innerhalb der status range einer Aktivität die Normtypen zweier *relevanter* status statements gleichzeitig „widersprechen".

Weiche Konflikte (O/P-Konflikte)

Ein weicher Konflikt repräsentiert die gleichzeitige Relevanz einer Verpflichtung und eines Rechts. Wenn eine Tätigkeit verpflichtend und auch erlaubt ist, dann gibt es aus logischer Sicht kein Problem, wenn diese Tätigkeit tatsächlich ausgeführt wird – der Konflikt „verschwindet". Wird die Tätigkeit dagegen nicht ausgeführt, dann kommt es zu einer Konfliktsituation. (Zu beachten ist, dass ein Recht zur Ausführung einer Aktivität die Möglichkeit der Unterlassung dieser Aktivität nicht ausschließt.)

Harte Konflikte (I/P - und I/O-Konflikte)

Harte Konflikte sind besonders kritisch, denn für sie gibt es keine Interpretation, die sie „verschwinden" lässt. Zwei verschiedene Varianten sind möglich:

1. **I/P-Konflikte**, also Konflikte die bei gleichzeitiger Relevanz eines Verbots und einer Erlaubnis einer Aktivität vorliegen, und
2. **I/O-Konflikte**, also Konflikte die bei gleichzeitiger Relevanz eines Verbots und einer Verpflichtung einer Aktivität vorliegen.

Beispiel 9 Harter Konflikt

Zwischen den beiden angegebenen Normspezifikationen liegt ein harter Konflikt vor, wenn die Variablen folgende Werte annehmen: `Article{beer}, CustomersAge{16..17}`

```
norm <i> <(Article = beer) AND (CustomersAge < 18)> <sell_article>
norm <p> <CustomersAge >= 16> <sell_article>
```

Sanktionsbasierte Konflikte

In dieser Konfliktklasse sind alle *sanktionsbasierten* Konflikte (auch *Konflikte 2. Ordnung* genannt) enthalten. Sie treten innerhalb der status range von Aktivitäten in zwei möglichen Varianten auf:

1. Es sind mindestens zwei *unterschiedliche Sanktionen* durch *unterschiedliche Rollen* möglich, und es ist deshalb nicht klar, welche Rolle nun mit ihrer entsprechenden Sanktionsaktivität sanktionieren darf bzw. muss.
2. Es gibt eine *eindeutige Sanktion*, aber sie kann von mindestens zwei *unterschiedlichen Rollen* über entsprechende Sanktionsaktivitäten ausgeführt werden und es ist deshalb unklar, welche Rolle für die Durchführung der Sanktionierung infrage kommt.

Konflikte der ersten Art treten immer dann auf, wenn es *gleichzeitig mindestens zwei Rollen* gibt, die eine bestimmte Aktivität auf *unterschiedliche Art sanktionieren* können. In diesen Fällen ist ohne die explizite Angabe einer zusätzlichen Information (Konfliktlösungsstrategie) nicht klar, welche Rolle nun tatsächlich für die Sanktionierung mit ihrer entsprechenden Sanktionsaktivität in Frage kommt.

Konflikte der zweiten Art treten immer dann auf, wenn für eine Aktivität zwar die *Sanktion eindeutig festgelegt* ist, es aber *gleichzeitig mindestens zwei Rollen* gibt, die sanktionieren können. In diesen Fällen ist ohne die explizite Angabe einer zusätzlichen Information (Konfliktlösungsstrategie) nicht klar, welche Rolle nun tatsächlich für die Sanktionierung mit ihrer entsprechenden Sanktionsaktivität infrage kommt.

Änderungskonflikte

Diese Konfliktklasse umfasst alle Konflikte, die mit der Spezifikation von Änderungsaktivitäten entstehen können; solche Konflikte werden änderungsbasierte Konflikte (oder auch *Konflikte 3. Ordnung*) genannt. Existieren *mindestens zwei Rollen*, die jeweils über eine entsprechende Änderungsaktivität *ein und dasselbe status statement* der status range einer bestimmten Aktivität auf *verschiedene Art und Weise in eine Änderung mit einbeziehen* können, dann ist nicht klar, wessen Änderungsaktivität tatsächlich eine Änderung herbeiführen soll beziehungsweise darf. Entscheidend für die resultierende status range ist demnach, welche Rolle Vorrang hat.

3.2 Strategien zur Konfliktauflösung

Im Folgenden werden drei elementare Strategien zur Konfliktlösung kurz beschrieben und es wird angemerkt, welche Konflikte mit durch ihre Anwendung gelöst werden können.

Strategie „Partielle Ordnung über Normen"

Diese Strategie ist nur für die Auflösung normbasierter Konflikte geeignet. Sie sieht vor, dass eine partielle Ordnung über die verschiedenen Norm-Typen definiert wird. Über diese Ordnung kann

festgelegt werden, welcher Norm-Typ im Falle eines Konflikts Vorrang hat. Beispielsweise könnte über eine entsprechende partielle Ordnung definiert werden, dass ein Verbot im Falle eines harten I/O- oder I/P-Konflikts Vorrang hat, für den Fall eines weichen O/P-Konflikts wird jedoch keine Aussage getroffen und der Konflikt bleibt bestehen (zur Auflösung dieses Konflikts könnte ergänzend einer weitere Strategie angewendet werden).

STRATEGIE „PARTIELLE ORDNUNG ÜBER ROLLEN“

Diese Strategie sieht vor, dass eine partielle Ordnung über die Rollen gelegt wird. Damit wird im Konfliktfall einer Rolle der Vorrang über eine andere Rolle gegeben. Weiche und harte Konflikte 1. Ordnung können immer dann gelöst werden, wenn sich die beiden involvierten Norm-Sanktions-Paare auf unterschiedliche Rollen beziehen. Konflikte 2. und 3. Ordnung können in jedem Fall durch eine geeignete partielle Ordnung über Rollen aufgelöst werden.

STRATEGIE „PARTIELLE ORDNUNG ÜBER HANDLUNGSAUFFORDERUNGEN“

Besteht innerhalb des status range einer Aktivität einer Rolle C ein Konflikt zwischen zwei „dependent statements“, die sich jeweils auf unterschiedliche Rollen A und B beziehen, so lässt sich dieser Konflikt auch so interpretieren, dass ein Agent als Inhaber der Rolle C gleichzeitig eine Handlungsaufforderung von einem Inhaber der Rolle A und einem der Rolle B erhält. Eine Strategie speziell zur Lösung solcher Konflikte ist die Festlegung einer partiellen Ordnung über die involvierten Handlungsaufforderungen. Grundsätzlich können Konflikte 1. Ordnung mit dieser Strategie gelöst werden, wenn die „konfliktverursachenden“ *status statements* vom Statustyp *dependent* sind. Konflikte 2. Ordnung können mit dieser Strategie nur gelöst werden, falls ein Befolgen oder Nichtbefolgen von Handlungsaufforderungen sanktioniert werden soll.

3.3 KOMBINATION VON KONFLIKTLÖSUNGSSTRATEGIEN

Um alle in einer Spezifikation enthaltenen Konflikte lösen zu können, müssen in der Regel mehrere Strategien angegeben werden. Aus Gründen der Determiniertheit muss allerdings klar sein, welche Strategie verwendet wird, falls mehrere zur Lösung eines Konflikts infrage kommen. Eine triviale Lösung dieses Problems ist die Festlegung einer totalen Ordnung über die zu verwendenden Strategien. Die resultierende geordnete Liste wird im Konfliktlöseprozess der Reihe nach solange abgearbeitet, bis sich eine Strategie findet, die in der Lage ist, einen zu beseitigenden Konflikt aufzulösen. Auf diese Weise ist jederzeit klar, welche Strategie zum Auflösen eines Konflikts verwendet wird.

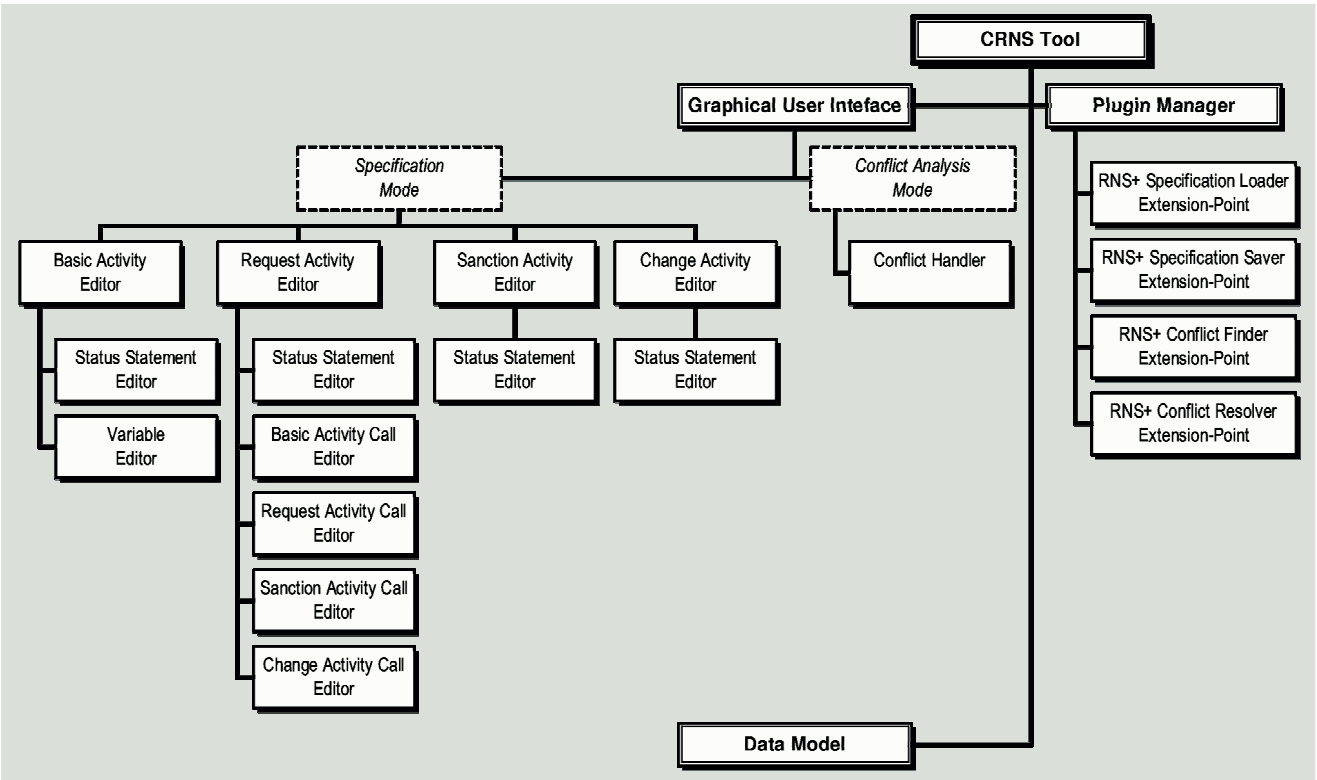
4 CRNS – EIN RNS+-BASIERTES SPEZIFIKATIONSTOOL

CRNS ist ein Tool, welches seine Benutzer bei der **RNS+**-basierten Spezifikation von autonomem Systemverhalten und bei der Konflikthandhabung (Identifikation und Auflösung) unterstützt. Im Folgenden wird ein Überblick über die Architektur von **CRNS** gegeben.

4.1 ARCHITEKTUR VON CRNS

Die Abbildung 1 gibt zunächst einen Überblick über die Architektur von **CRNS**. Anschließend werden zentrale Komponenten von **CRNS** kurz vorgestellt.

Abbildung 1 Überblick über die Architektur von CRNS



Der Kern von **CRNS** besteht aus folgenden Komponenten:

- **Das Datenmodell** ist die interne Repräsentation einer **RNS+** Spezifikation.
- **Die grafische Benutzeroberfläche** ermöglicht das Laden und Speichern, sowie die Eingabe/Abänderung einer **RNS+** Spezifikation (*Spezifikations-Modus*). Außerdem ist eine Konfliktanalyse über Konfliktfindungs- und Konfliktlösungsstrategien möglich (*Analyse-Modus*).
- **Der Plugin Manager** ist für die Registrierung und Verwaltung von zusätzlichen Softwaremodulen (sog. Plugins) verantwortlich, die dem Tool Lade- und Speicherroutinen, sowie Konfliktfindungs- und Konfliktlösungsstrategien zur Verfügung stellen.

DIE GRAFISCHE BENUTZEROBERFLÄCHE

Die Komponenten der Benutzeroberfläche von **CRNS** lassen sich nach Spezifikations-Modus (Abbildung 2) und Konfliktanalyse-Modus (Abbildung 3) ordnen.

Im Hinblick auf den *Spezifikations-Modus* sind folgende zentrale Komponenten zu unterscheiden:

- **Der „Variable Editor"** erlaubt das Editieren einer Variable, d.h. über ihn können ihr Name und ihr Wertebereich festgelegt werden.
- **Der „Status Statement Editor"** erlaubt das Editieren eines status statement. Da alle Aktivitätstypen eine status range besitzen ist er aus allen Aktivitätseditoren erreichbar.
- **Der „Basic Activity Editor"** erlaubt das Editieren der Variablenliste und des status range der Aktivität.
- **Der „Request Activity Editor"** erlaubt u.a. das Editieren von Listen von Agenten, an die eine Handlungsaufforderung gerichtet werden kann. Des Weiteren kann über entsprechende „Activity Call" Editoren die Aufforderung bzgl. der (Nicht-)Ausführung einer beliebigen Aktivität festgelegt werden.
- **Der „Sanction Activity Editor"** erlaubt u.a. das Editieren von Listen von Agenten, die sanktioniert werden können.
- **Der „Change Activity Editor"** erlaubt u.a. das Editieren von Listen von Rollen, welche die betroffenen Aktivitäten enthalten, sowie von Listen der zu ändernden Aktivitäten selbst.

Des Weiteren kann spezifiziert werden, ob ein status statement hinzugefügt, gelöscht, oder durch ein anderes ersetzt werden soll.

Im *Konfliktanalyse-Modus* kann in der sich im Speicher befindlichen **RNS+** Spezifikation mit Hilfe der „Conflict Handler" Komponente nach Konflikten gesucht werden. Anschließend ist die Anwendung von Konfliktfindungs- und Konfliktlösungsstrategien möglich; welche Strategien dabei zur Verfügung stehen, kann dem „Conflict Handler" durch eine entsprechende Eingabe über die Benutzerfläche bekanntgegeben werden.

Abbildung 2 CRNS im Spezifikations-Modus

Rechts ist ein „Request Activity Editor" für die Aktivität `interdict_catalog_entry_insertion` der Rolle `platform_mg` abgebildet. [Fortsetzung nächste Seite]

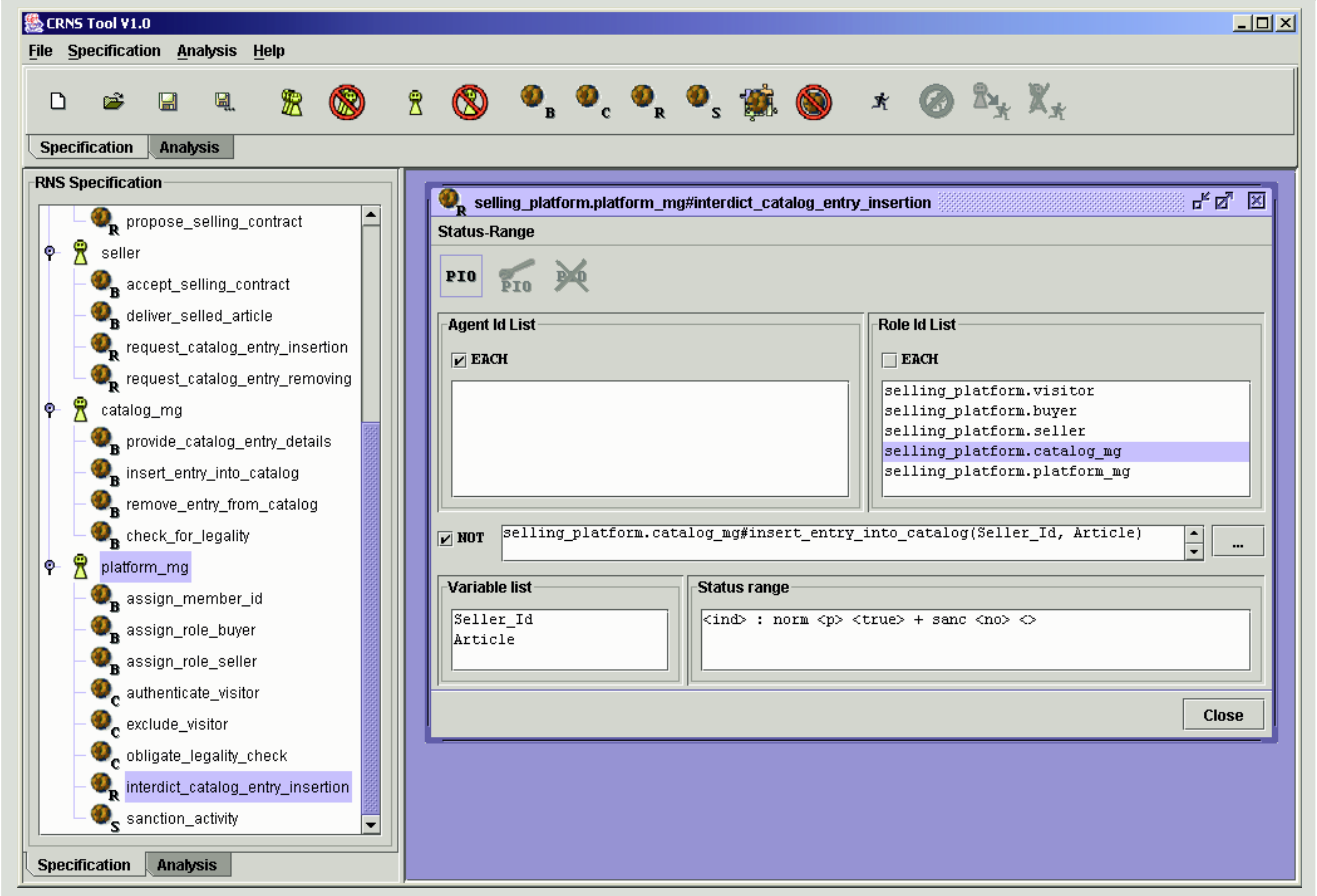
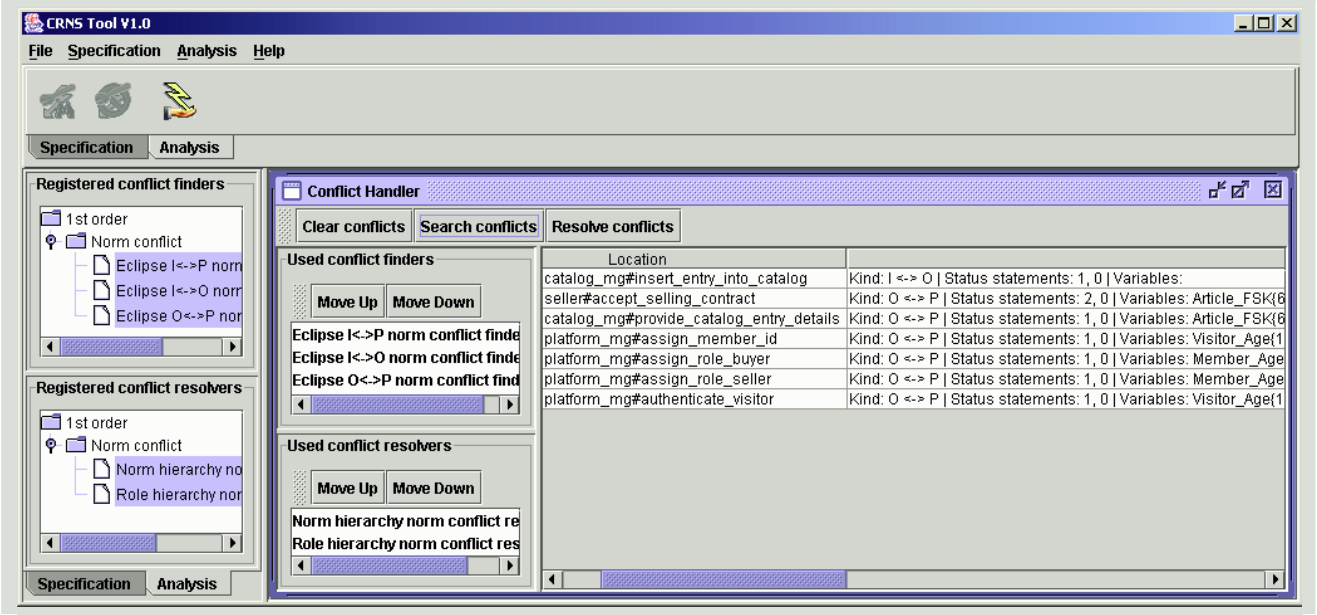


Abbildung 3 CRNS im Analyse-Modus

Im abgebildeten „Conflict Handler" Fenster werden sechs Konflikte angezeigt (berechnet mittels **ECL^{i}PS^e**). Diese wurden mittels der Plugins berechnet, welche für das Auffinden normbasierter Konflikte zuständig sind.



Der Plugin Manager

Der Plugin Manager registriert und verwaltet Softwaremodule (sog. *Plugins*), welche **CRNS** über entsprechende Schnittstellen (sog. *Extension-Points*) um bestimmte Funktionalitäten bzw. Strategien erweitern. Folgende Extension-Points sind für Plugins definiert:

- **„RNS+ Specification Loader"** als Schnittstelle für ein Plugin, das eine Laderoutine zum Lesen einer **RNS+** Spezifikation zur Verfügung stellen möchte.
- **„RNS+ Specification Saver"** als Schnittstelle für ein Plugin, das eine Speicherroutine zum Schreiben einer **RNS+** Spezifikation zur Verfügung stellen möchte.
- **„RNS+ Conflict Finder"** als Schnittstelle für ein Plugin, das in einer **RNS+** Spezifikation nach Konflikten eines bestimmten Typs zu suchen.
- **„RNS+ Conflict Resolver"** als Schnittstelle für ein Plugin, das gefundene Konflikte eines bestimmten Typs in einer **RNS+** Spezifikation auflösen kann.

Für die Installation eines Plugins müssen dessen Programmkomponenten vor dem Programmstart lediglich in ein bestimmtes Verzeichnis kopiert werden, weshalb **CRNS** sehr leicht konfigurierbar ist.

4.2 Konflikthandhabung – Normbasierte Konflikte & ECL^{i}PS^e

CRNS automatisiert vollständig die Handhabung der wichtigen Klasse der (weichen und harten) normbasierten Konflikte. Die Identifikation dieser Konflikte erfordert einen nichttrivialen Mechanismus, der es erlaubt herauszufinden, unter welchen Voraussetzungen zwei beliebige Bedingungsterme *gleichzeitig* erfüllt sein können, d.h. für welche Belegungen der in ihnen enthaltenen Variablen die Interpretation beider booleschen Ausdrücke den Wert „wahr" ergibt. Genau diesen Mechanismus stellt **ECL^{i}PS^e** über die `fd`-Bibliothek zusammen mit der `propia`-Bibliothek zur Verfügung. **CRNS** verwendet deshalb Plugins, welche die Java-Schnittstelle der logischen Programmiersprache **ECL^{i}PS^e** („Constraint Logic Programming System") verwenden (siehe z.B. [1,4]).

5 Schlussbetrachtungen

Die Sprache **RNS+** ermöglicht, wie kein anderer derzeit verfügbarer Formalismus [2,5,10,11,14], eine sehr präzise und ausdrucksstarke Spezifikation von autonomen Verhalten von Softwareagenten. Weiterhin liegt mit **CRNS** liegt ein leistungsfähiges und komfortables Tool zur komfortablen Erstellung von **RNS+**-basierten Spezifikationen vor. Ein herausragendes Merkmal dieses Tools ist seine umfangreichen Unterstützung der automatisierten Handhabung von normbasierten Konflikten.

Die präzise anwendungsspezifische Spezifikation von autonomen Systemverhalten stellt eine zentrale Herausforderung dar an das agentenorientierte Software Engineering und – mit zunehmender Bedeutung von autonomen Softwaresystemen – darüber hinaus für das Software Engineering im Allgemeinen. Ein breiter Einsatz von agentenorientierter bzw. autonomer Software in kommerziellen Anwendungen hängt deshalb in besonderem Maß von der Verfügbarkeit von Formalismen und Tools ab, die eine präzise Spezifikation von Autonomie als Softwareeigenschaft ermöglichen. **RNS+** und **CRNS** wurden in Hinblick auf diesen Bedarf entwickelt.

Literatur

[1] **ECL^{i}PS^e (2003)**: http://www.icparc.ic.ac.uk/eclipse/

[2] **Barbuceanu M., Gray T., Mankovski S (1999):** *The role of obligations in multiagent coordination.* Journal of Applied Artificial Intelligence, Vol. 13(2/3), 11-38.

[3] **Biddle B.J., Thomas E.J. (Eds.) (1966):** *Role Theory: Concepts and Research.* John Wiley & Sons, Inc.

[4] **Cheadle A., Harvey W., Shen K., Sadler A., Wallace M., Schimpf J. (2002):** *ECL^{i}PS^e: An Introduction,* London, Imperial College, IC-Parc, **ECL^{i}PS^e** Group.

[5] **Dignum F. (1999)**: *Autonomous Agents with Norms. Artificial Intelligence and Law, Vol. 7, 69-79.*

[6] **Giunchiglia F., Odell J., Weiß G. (Eds.) (2003):** *Agent-oriented Software Engineering III.* LNCS Vol. 2585, Springer-Verlag.

[7] **Hewlett-Packard, Adaptive Enterprise Initiative (2003)**: http://www.hp.com/large/globalsolutions/ai.html?jumpid=go/adaptive

[8] **IBM, Autonomic Computing Initiative (2003):** http://www.research.ibm.com/autonomic/

[9] **Jennings N.R. (2000):** *On agent-based software engineering. Artificial Intelligence,* Vol. 117(2), 277-296.

[10] **Lopez y Lopez F., Luck M., d'Inverno M. (2002):** *Constraining autonomy through norms*. Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'02, pp. 674-681).

[11] **Lupu E. and Sloman M. (1997):** *Towards a Role based Framework for Distributed Systems Management*. Journal of Network and Systems Management, Vol. 5(1), 5-30.

[12] **Microsoft, Dynamic Systems Initiative (2003)** : http://www.microsoft.com/presspass/press/2003/mar03/03-18dynamicsystemspr.asp

[13] **Nickles M., Rovatsos M., Weiß G. (2003):** *A Schema for Specifying Computational Autonomy,* Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02, pp. 82-95), LNCS Vol. 2577, Springer-Verlag.

[14] **Pacheco O., Carmo J. (2002):** *A Role based Model for the Normative Specification of Organized Collective Agency and Agents Interaction*. Journal of Autonomous Agents and Multi-Agent Systems, Vol. 6(2), 145-184.

[15] **Sun, N1 Initiative (2003):** http://wwws.sun.com/software/solutions/n1/index.html

[16] **Weiß G. (2002)**: *Agent Orientation in Software Engineering*. Knowledge Engineering Review, Vol. 16(4), 349-373.

[17] **Weiß G, Rovatsos M., Nickles M. (2003):** *Capturing Agent Autonomy in Roles and XML,* Aproceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), Melbourne, Australia.

[18] **Wooldridge M., Weiß G., Ciancarini P. (Eds.) (2002):** *Agent-orient oriented Software Engineering II.* LNCS Vol. 2222, Springer-Verlag.