Ein priorisierbares Informationsmodell für die Verwaltung von Dienstkonfigurationsdaten

Ralf Döring
ralf.doering@tu-ilmenau.de

Technische Universität Ilmenau

Zusammenfassung Die Konfiguration von Dienstinstanzen stellt einen wesentlichen Aspekt bei einer dienstorientierten Betrachtung verteilter Systeme dar. Trotz der immensen Bedeutung besteht auf diesem Sektor noch großer Nachholebedarf. Versionierung von Konfigurationsdaten und Verringerung der Redundanz der darin enthaltenen Daten stellen wesentliche Aspekte bei der Verwaltung von Dienstkonfigurationsdaten dar. Aus diesem Grunde wurde ein Informationsmodell unter Berücksichtigung dieser Anforderungen entwickelt. Diese Modell wird dann um eine prioritätsbasierte Verwaltung unterschiedlicher Konfigurationsquellen erweitert.

1 Einführung

In der heutigen vernetzten Welt spielt die Nutzung im Netzwerk verteilter Dienste eine immer grössere Rolle. Verschiedene, zum Teil miteinander konkurrierende Frameworks, die solche Dienste im Rahmen sogenannter *Web-Services* allgemein verfügbar machen, sprechen für eine weiter wachsende Bedeutung einer dienstorientierten Betrachtungsweise von Netzwerken, bei der die kommunikationsorientierten Aspekte in den Hintergrund treten. In den letzten Jahren wurden grosse Anstrengungen unternommen, um allgemein akzeptierte Spezifikationen für das Auffinden und Benutzen von Diensten in heterogenen Netzen zu erstellen. Eines haben alle diese existierenden Technologien gemeinsam: sie betrachten im wesentlichen welche *Art* Dienstleistung von einem *Dienstbenutzer wie* in Anspruch genommen werden kann. Dies ist sicher ein wichtiger, wenn nicht der ausschlaggebende Punkt in einem dienstorientierten Modell. Aus der Sicht desjenigen, der eine Vielzahl von Diensten zu administrieren hat, ist jedoch der Punkt, *wie* das *Verhalten* des *Dienstes* bei der *Diensterbringung* zu beeinflussen ist. Dieser Punkt wird landläufig als *Dienstkonfiguration* bezeichnet, auch wenn dieser Begriff in mehreren, miteinander verwandten aber doch unterschiedlichen, Bedeutungen verwendet wird (vgl. z.B. [6, 10, 8]). Dabei steht folglich die Frage, *wie* ein Dienst seine Dienstleistung erbringt, im Vordergrund.

Zwar existieren bereits Ansätze und Bestrebungen für ein einheitliches Modell zur Dienstintrospektion (vgl. z.B. [3, 9, 1, 3, 2, 4]), jedoch lassen auch diese Technologien und Initiativen wesentliche Punkte, wie die Versionierung von Konfigurationsdaten, aussen vor.

2 Das Basismodell

Das den weiteren Ausführungen zugrundeliegende Modell wurde bereits in [5] entworfen und soll hier in einer leicht modifizierten Version noch einmal kurz vorgestellt werden. Dabei flossen die folgenden Kriterien in den Entwurf eines grundlegenden Informationsmodells ein:

1. Eine Versionierung der Konfigurationsdaten ist unbedingt zu berücksichtigen.
2. Redundanzen bei der Konfiguration gleichartiger und unterschiedlicher Dienstypen sind möglichst zu vermeiden.

Die beiden aufgeführten Punkte resultieren im wesentlichen aus Erfahrungen in der Praxis. Gängige Varianten der Dienstkonfiguration (z.B. über Konfigurationsdateien) haben in der Regel keine oder nur über externe Mechanismen realisierbare Unterstützung für mehrere Versionen einer Dienstkonfiguration. Aus Sicht eines Administrators, der die Parameter eines Dienstes ändern möchte, ist somit ein explizites Sichern der aktuellen Konfiguration eines Dienstes notwendig, um beim Fehlschlagen der Parameteränderung zur vorherigen Version zurückkehren zu können. Wurde diese vorherige Version jedoch nicht *„per Hand“* gesichert, ist ein „Zurück zum letzten Zustand“ nicht einfach. Durch ein Konfigurationsmodell, welches eine Versionierung der Dienstkonfiguration unterstützt, lässt sich die Relevanz dieser Fehlerquelle nahezu ausschliessen. Eine als „funktionierend“ (im Sinne der Spezifaktion des Dienstes) bekannte Konfiguration erhält ein eigenes Kennzeichen, die *Revisionsnummer*. Darunter ist eine Zusammenstellung aller Parameter des Dienstes zu verstehen, die den aktuell verwendeten Initialzustand des Dienstes wieder herbeiführen können. Änderungen an einer bereits existierenden Version sollten somit zu einer neuen Version führen und die Zusammenstellung existierender Parametersets zu Konfigurationsversionen nicht beeinträchtigen.

Eine weitere potentielle Fehlerquelle stellen Redundanzen in der Konfiguration verschiedener Dienstinstanzen dar. Muss ein Parameter geändert werden, so ist darauf zu achten, diesen Parameter in der Konfiguration aller Instanzen zu ändern. Wenn jede Instanz eine eigene Kopie der Konfigurationsdaten erwartet, so ist einzusehen, das bei diesen Änderungen leicht eine Instanz „vergessen“ werden kann. Für gemeinsam genutzte Parameter ist somit eine einmalige Speicherung von Vorteil, da dann die Änderung nur einmal durchgeführt werden kann. Daraus lässt sich ableiten, das die Konfiguration einer Instanz prinzipiell in zwei Mengen unterteilt werden kann:

1. Die Menge der gemeinsamen Parameter, die mit anderen Instanzen geteilt werden
2. Die Menge der „privaten“ Parameter einer Instanz.

Resultierend aus diesen grundlegenden Anforderungen wurde das in Abbildung 1 gezeigte Informationsmodell für die Beschreibung von Dienstkonfigurationen abgeleitet. Dieses Modell ist als Baum realisiert, welcher aus Knoten zur

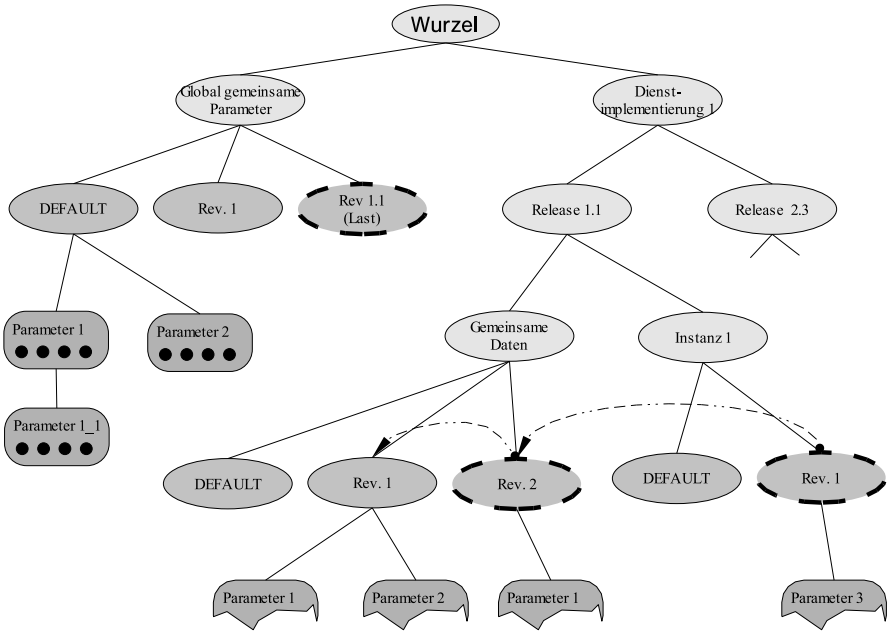


Abbildung 1. Das grundlegende Modell

Strukturierung der verwalteten Informationen und aus Knoten zur Speicherung der eigentlichen Konfigurationsdaten besteht. Es wird davon ausgegangen, dass die direkten Kindknoten jedes Knotens eindeutige Namen besitzen und damit eindeutig unterscheidbar sind. Die Forderungen nach Versionierung der Konfigurationsparameter und der Verringerung von Redundanzen werden durch die Struktur des Baumes berücksichtigt.

Basis der Dienstkonfiguration sind die als Blätter des Baumes zu findenden *Parameterknoten*. Ein Parameter beschreibt mit den enthaltenen Attributen einen bestimmten Aspekt des Verhaltens der Dienstinstanz. Die gesamte, für die Wiederherstellung eines gewünschten Verhaltens einer Dienstinstanz notwendige, Konfigurationsinformation kann aus mehreren Parametern zusammengesetzt sein und wird dann als *Parameter-Set* bezeichnet.

Der Baum lässt sich von der Wurzel aus betrachtet in zwei unterschiedliche Bereiche unterteilen. Der unter dem Knoten *„Global gemeinsame Parameter“* angesiedelte Teilbaum verwaltet globale Parameter, die für alle Arten von Diensten, unabhängig vom Diensttyp, gelten sollen. Die weiteren unterhalb der Wurzel angeordneten Teilbäume beschreiben jeweils *genau* eine Implementierungsvariante eines Dienstes. Sie sind hier als *Dienstimplementierung* bezeichnet. Ein Beispiel wären jeweils separate Teilbäume für den *HTTP-Server Apache* und den *HTTP-Server Roxen*. Jede dieser Implementierungen kann in unterschiedlichen Versionen der zugrundeliegenden Software vorliegen (z.B. Apache 1.3.38, Apache 2.0.25,...). Da sich die Art und Weise der Konfiguration zwischen ver-

schiedenen Softwareversionen des selben Dienstes unterscheiden kann, werden konsequenterweise *Release-Knoten* eingeführt, um diesen Strukturierungsaspekt im Baum abzubilden. Diese korrespondieren mit einem „Release“ der Dienstimplementierung.

Zur Verringerung von Redundanzen können gemeinsame Daten, die für verschiedene Instanzen unterhalb des gleichen Release-Knotens gelten sollen, unterhalb des Knotens *Gemeinsame Daten* angesiedelt werden. Auf der gleichen Ebene befindet sich auch eine beliebige Anzahl von *Instanzknoten*. Jeder Teilbaum unterhalb eines solchen Knotens repräsentiert die Konfigurationsdaten für *genau* eine Instanz eines Dienstes.

Als letzter Strukturknotentyp finden sich sogenannte *Revisionsknoten* im Baum wieder. Diese dienen der Modellierung der Versionierung der Konfigurationsdaten. Sie repräsentieren eine Menge von Konfigurationsparametern, die zusammen eine gewünschte (initiale) Konfiguration ergeben. Zur weiteren Verringerung von Redundanzen kann eine *Defaultrevision* existieren. Alle anderen Revisionen der Dienstkonfiguration speichern dann nur die gegenüber dieser Defaultrevision veränderten Parameter. Desweiteren kann eine Revision der Konfiguration als *aktuelle* oder *letzte* Konfiguration markiert werden. Diese wird beim Start einer Instanz als Initialkonfiguration verändert. In Abbildung 1 sind diese Knoten durch eine gestrichelte Umrandung markiert. Die Existenz einer Ordnungsrelation zwischen den Revisionknoten wird vorausgesetzt.

Als zusätzliches Element existieren Referenzen zwischen *Revisionsknoten*. Damit ist es möglich, Sachverhalte wie *„. . . Revision 1.5 bezieht sich auf Revision 1.1. . . “* zu formulieren. Damit muss eine Revision nur noch die Konfigurationsparameter enthalten, die gegenüber der referenzierten Version verändert wurden.

Die Suche nach einem bestimmten Konfigurationsparameter für eine Dienstinstanz wurde bereits in [5] beschrieben und soll hier nicht noch einmal aufgeführt werden.

3 Erweiterung um Prioritäten für Konfigurationsquellen

Das vorgestellte Basismodell ermöglicht bereits eine Unterscheidung in *global* geltende, für die *Gruppe von Instanzen gleichen Release-Standes* geltende sowie für *eine einzelne* Instanz geltende Konfigurationsparameter sowie eine Versionierung von Konfigurationen. Weitergehende Aussagen über die Verwendung und Priorisierung von Parametern sind damit nicht möglich. Instanz- oder dienstspezifische Konfigurationen, die einzelnen Nutzern oder Nutzergruppen zugeordnet sind, sind ebenfalls nicht realisierbar.

In der Praxis finden sich häufig auch Aussagen wie „. . . der Wert dieses Konfigurationsparameters darf von einem Benutzer nur verändert werden, wenn vom Administrator keine Vorgaben für diesen Parameter gemacht wurden. . . “. Konfigurationsparametern kann somit eine gewissen *Priorität* zugeordnet werden. Existierende Dienstimplementierungen stellen ihre Initialkonfiguration meist aus verschiedenen Quellen zusammen, um administrative Vorgaben und die Einstel-

lungen der Dienstnutzer zu berücksichtigen. Diese verschiedenen Quellen können z.B. nutzer-, gruppen- und standortspezifische Daten umfassen. Die Priorisierung von Parametern aus diesen unterschiedlichen Quellen ist dann in der Regel an die Quelle das Parameters gekoppelt und in der Dienstimplementierung fest verankert.

Für ein universelles Dienstkonfigurationsmodell ist die Modellierung eines analogen Verhalten wünschenswert. Allerdings sollte hier die Auswahl und Priorisierung der Parameter *nicht* der Dienstimplementierung obligen, sondern durch das Konfigurationssystem selbst realisiert werden.

Als Möglichkeit, dieses Verhalten im Basismodell zu integrieren, werden die folgenden Veränderungen aufgenommen:

1. Für die Festlegung der Konfiguration einer Dienstinstanz können mehrere *Konfigurationsquellen* herangezogen werden.
2. Jede Konfigurationsquelle wird mit einer eindeutigen *Priorität* versehen. Die Prioritäten aller für die Bildung einer Konfiguration verwendeten Quellen müssen voneinander unterschiedlich sein.
3. Die einzelnen Parameter einer Quelle erhalten eigene Prioritäten, die jedoch mit der Priorität der Quelle identisch sein können. Es können in einer Quelle sowie zwischen verschiedenen Quellen Parameter gleicher Priorität existieren.
4. Die Existenz einer Ordnungsrelation zwischen den Prioritäten wird vorausgesetzt.

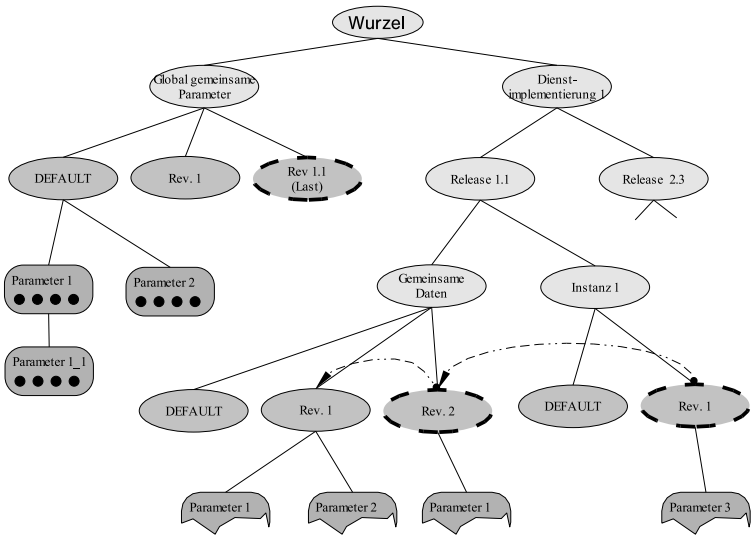
Anhand der oben genannten Kriterien ist eine Erweiterung des erläuterten Informationsmodells möglich. Die *Priorität* einer Quelle oder eines Parameters kann im einfachsten Fall ein Ganzzahlwert sein.

Zur Realisierung werden die Konfigurationsparameter nach ihrer *Quelle* zusammengestellt. Jede Quelle bildet in diesem Sinne ein abgeschlossenes, nach dem Informationsmodell aus Abbildung 1 strukturiertes Konfigurationssystem ab. Als Beispiel können Quellen die Konfiguration einer Dienstinstanz für einen bestimmten Nutzer oder eine bestimmte Nutzergruppe abbilden.

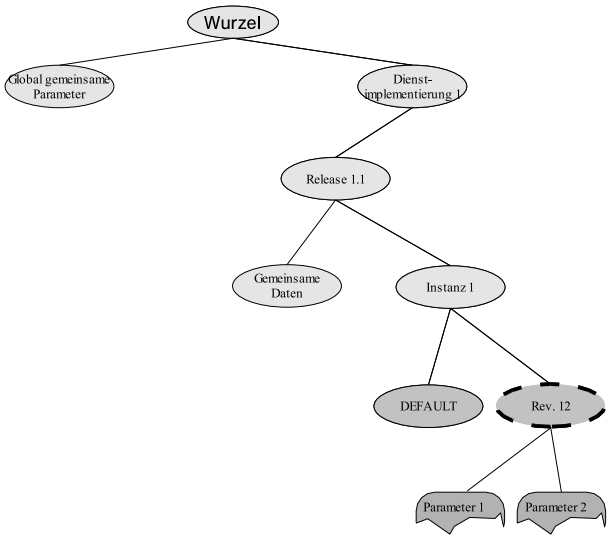
Abbildung 2 zeigt ein Beispiel, bei dem sich die endgültige Konfiguration aus zwei Konfigurationsquellen für die global für alle gültige Konfiguration, sowie der Konfiguration für einen speziellen Nutzer zusammensetzen soll.

In einem ersten Schritt wird für jede Quelle anhand des Parametersuchalgorithmus ein vollständiges Parameterset aller enthaltenen Parameter zusammengestellt und in einem temporären Revisionsknoten abgelegt. Diese Aktion ist somit eine Umkehr der redundanten Parameterspeicherung. Der temporäre Revisionsknoten enthält dann alle für die Quelle bekannten Konfigurationsparameter für eine gegebene Dienstinstanz, inklusive der aus den Default, gemeinsamen und globalen Teilbäumen gewonnen Parameter.

In einem zweiten Schritt werden alle Parameterknoten der temporären Revision mit einer Priorität versehen. Unter einer Priorität soll hier ein ganzahliger Wert verstanden werden. Prinzipiell kann dabei jeder Parameter jeder Quelle eine unterschiedliche Priorität erhalten oder die Priorität der Parameter pro



(a) Globale Konfiguration

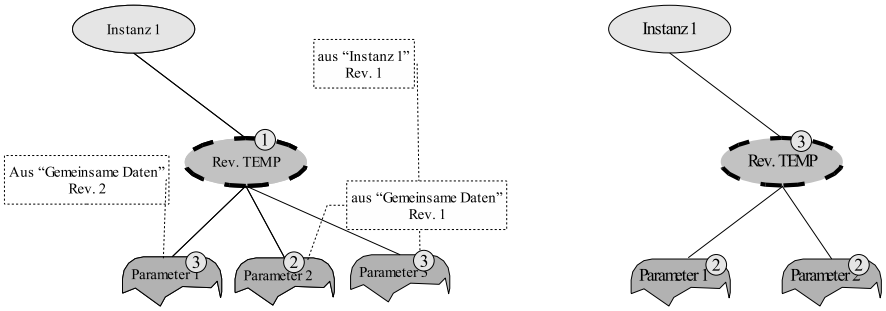


(b) Nutzerspezifische Konfiguration

Abbildung 2. Zusammenstellung von Konfigurationsquellen

Quelle konstant sein. Für die Vergabe der Prioritäten können verschieden Kriterien, wie standortbezogene oder dienstimplementierungsspezifische Vorgaben, herangezogen werden. Da eine generelle Aussage über die Art und Weise der Festlegung weder möglich noch notwendig erscheint, liegt folglich die Bestimmung der Prioritäten ausserhalb des Kontextes dieses Beitrages

Zusätzlich wird der temporären Revision jeder Quelle eine zusätzliche Priorität zugeordnet, die von den Prioritäten der jeweils anderen Quellen unterschiedlich ist. Dabei kann einfach die Priorität der Quelle übernommen werden. Dies wird für die Auflösung von Konflikten benötigt. Das Ergebnis dieser Operation ist, ausgehend von den Beispielen aus Abbildung 2 in Abbildung 3 für *Instanz 1* gezeigt. Die angezeichneten Prioritäten wurden beispielhaft vergeben.



(a) Globale Konfiguration

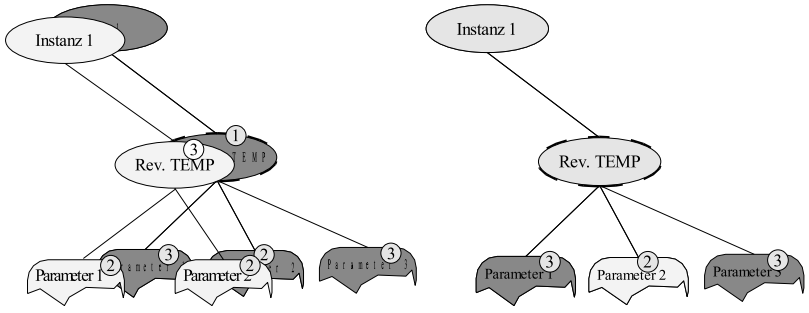
(b) Nutzerspez. Konfiguration

Abbildung 3. Temporäre Revisionen für Instanz 1 mit Prioritäten

In einem dritten Schritt werden die temporären Revisionen der einzelnen Quellen in Deckung gebracht und zunächst als komplette Teilbäume in absteigender Ordnung der Prioritäten sortiert. Dieser Schritt ist notwendig, um festzulegen, welcher Parameter bei gleicher Parameterpriorität nach der Anordnung entsprechend dem nächsten Schritt ausgewählt wird.

In einem letzten Schritte werden schliesslich alle deckungsgleichen Parameter ebenfalls in absteigender Folge ihrer Prioritäten sortiert und alle „verdeckten" Parameter entfernt. Parameter mit gleicher Priorität werden in der selben Reihenfolge angeordnet wie die zugehörigen temporären Revisionen.

Das Ergebnis der Sortierung der letzten beiden Schritte ist in Abbildung 4 dargestellt. Durch die vorgenommene Sortierung anhand von Revisions- und Parameterprioritäten lässt sich eine eindeutige und wiederholbare Zusammenstellung eines Parametersets aus unterschiedlichen Quellen erreichen.



(a) Sortierung nach Quellen

(b) Sortierung nach Parametern

Abbildung 4. Ergebnis

4 Zusammenfassung und Ausblick

Das bereits vorhandende Modell zur Speicherung von Konfigurationsparametern nach Abschnitt 2 ist zunächst nicht gut geeignet, um Konfigurationssets aus unterschiedlichen Quellen zu beziehen und eine resultierende Konfiguration abzuleiten. Der vorgeschlagene Weg der Strukturierung, Priorisierung und Sortierung der Konfigurationsparameter ermöglicht die eindeutige Zusammenstellung von Parametersets aus einer beliebigen Anzahl von Quellen.

Für eine Anwendbarkeit in der Praxis muss in einem zukünftigen Schritt die Vergabe von Prioritäten der Konfigurationsquellen und -parameter betrachtet werden. Hierzu ist für ein auf dem vorgeschlagenen Informationsmodell aufbauenden Konfigurationssystem die formale Beschreibung zu verwendender Konfigurationsquellen Voraussetzung. Zusätzlich muss die Beschreibungsmöglichkeit von Parameterprioritäten in Abhängigkeit von der verwendeten Quelle betrachtet werden.

Literaturverzeichnis

[1] : *Common Information Model*. DMTF Specification. – URL `http://www.dmtf.org/spec/cims.html`

[2] : *Desktop Management Taskforce*. – URL `http://www.dmtf.org`

[3] : *Directory Enabled Networks Standards*. – URL `http://www.dmtf.org/spec/denh.html`

[4] : *Web Based Enterprise Managements Standards*. – URL `http://www.dmtf.org/spec/wbem.html`

[5] Döring, Ralf: Towards a Service Configuration Model. In: *Proc. 8th International Netties Conference*. [11]

[6] Hegering, Heinz-Gerd ; Abeck, Sebastian ; Neumair, Bernhard: *Integriertes Management vernetzter Systeme.* dpunkt-Verlag, 1999

[7] Kampman, Kevin ; Kampman, Christina: *All about Network Directories: Understanding Directory Services and Business Applications.* Wiley Computer Publishing, 2000

[8] Mann-Rubinson, Teres C. ; Terplan, Kornel: *Network Design: Management and Technical Perspectives.* CRC Press, 1998

[9] Strassner, John: *Directory enabled networks.* Macmillan Technical Publ., Indianapolis, Ind., 1999. – ISBN:1-578-70140-6

[10] Subramanian, Mani: *Network Management: An introduction to principals and practice.* Addison-Wesley, 2000

[11] TU Ilmenau (Veranst.): *Proc. 8th International Netties Conference.* 2002