セマンティックウェブサービスのためのタスク処理言語
Task Processing Language for Semantic Web Servicing

小出 誠二 Seiji Koide
島田 紀一 Norikazu Shimada

株式会社ギャラクシーエクスプレス
Galaxy Express Corporation

Local variables, the scope, and data flow from inputs of Web Services to outputs are introduced into the OWL-S 1.1 specification. However, these new features are not useful for agents to discover and compose Web Services. Agents must know the meanings of services through interpreting service parameters, i.e., inputs, outputs, preconditions, and effects (IOPEs). The subsumption of IOPEs helps agents to obtain the meanings, but understanding local variables and the scope is intricate for agents. The scope of variables and data flow is not what agents interpret but what agents generate as a result of web composing. Furthermore, it is very laborious for ones to describe tasks with the OWL-S expression. Therefore, based on Scheme and SWCLOS, a Semantic Web processor on top of CLOS, we design a task language that describes tasks and executed by agents. A task is captured as an ensemble of service, simple process, and complex process or atomic processes in OWL-S. Agents can obtain the meanings of services with subsumption of IOPEs of tasks, and composes them in order to accomplish given goals. Agents may perform those tasks directly, or may compile the task expressions into pieces of OWL-S code that includes descriptions of local variables and the scope.

1. はじめに

セマンティックウェブサービス(SWS)のためのマークアップ言語 OWL-S 1.1 では，新たに局所変数とスコープ概念が導入された．スコープとは，変数の参照が生じうるようなプログラムの空間的なあるいは文脈的な領域を指し示すものであり，それに対してエクステントとは，参照が生じ得るような時間間隔を指すものである[Steele 1990]．しかし，スコープとかエクステントはオントロジーにおいてマークアップされるものというよりは，プログラム記述に現れるものである．1.1 において「プロセスは実行されるプログラムではない」，「それはクライアントがサービスとインタラクトする方法の仕様である」と述べられたことは 1.0 に比べて進歩と考えるが，それにもかかわらず 1.1 で提出された仕様の中には，「インタラクトする方法の仕様」というよりも，プログラム記述と言うべき部分がある．OWL-S 1.1 にはプログラム仕様記述に相当する部分とプログラム記述に相当する部分が混在している．

我々は OWL-S で記述されたコンテンツはタスクオントロジーであると考えているが，同時にそれはプログラムそのものであるという立場に立っていた．そのため，エージェントに関する先の考察において[小出 2004]，「はたして，現在（2004 年当初）の OWL-S 仕様で変数の導入が可能になるのか，OWL-S 仕様を拡張して，実用的な範囲でスコープとエクステントの問題を解決できるのであろうか」と指摘した．しかし報告時の研究会における議論と我々のその後の経験により，現在では OWL-S はプログラムではなく，プログラム作成のための仕様でなければならないと認識するに至った[小出 2005]．

エージェントがウェブサービスの意味を発見し，それを組み合わせるさいに，局所変数やスコープが役に立つとは思えない．ウェブサービスの前提条件，入出力，効果(IOPE)のパラメータは，ウェブサービスのタキソノミー（外延）を支える内包である．エージェントはそれらの IOPE を見て，素朴にはパラメータの包摂概念を用いてサービスの意味を発見し，精巧には前提条件や条件付き効果の記述を見て，可能な組み合わせを発見することができる．しかし局所変数の束縛を見て，データフローを追跡することでエージェントはサービスの意味を理解することができるであろうか．我々は，ウェブサービス用エージェントにとって局所変数やスコープはそれを解釈するものではなく，エージェントの一部である計画プログラムがウェブ合成において生成する結果として，エージェントの行動に現れるものと考える．

本報告では，計画部，実行部，記憶部，インタフェースからなるエージェントがウェブサービスを呼び出すためのタスク言語を，設計する．類似研究に McDermott と Dou による Web-PDDL[McDermott 2002]や PDDAML があるが，前者は異質なオントロジーをマージするためのものであり，後者は PDDL と DAML 表記の双方向翻訳を目的としており，いずれもエージェントによって実行されるタスク言語ではない．PDDL とは AI 計画システムコンペティション(AIPS)のための公式タスク表現言語であるが，ウェブサービス実行のためのものではない．ここで提案するタスク表現言語は，それによってウェブサービス用エージェントが記述されたコンテンツから，ウェブサービス発見と合成を行い，実行するものである．タスクを組み合わせるときにエージェントが局所変数やスコーピングを理解する必要はなく，それはこのタスク言語記述の解釈実行時にエージェントの動作中に暗に存在するか，あるいはそれをコンパイルしたときに結果としてプログラムテキスト中に局所変数やスコーピングが現れるものと考える．コンパイル結果であるプログラムテキストは，このタスク言語でも表現できるが OWL-S 1.1 でも表現できる．

ここで想定するタスク言語は，本質的には OWL-S の記述能力を超えるものではないが，OWL-S を用いずに，それに代わるタスク記述言語を設計する理由は，タスクオントロジー記述における人間の労力軽減と解釈実行系設計簡素化のためである．これまでの我々の経験から，タスクオントロジーを OWL-S 記述するのは大変な作業であることがわかっているし，その解釈実行系を設計するのも大変な仕事である．たとえば，OWL-S 記述で

連絡先：小出誠二，（株）ギャラクシーエクスプレス，港区浜松町 1-18-16，Fax03-5733-7190，koide@galaxy-express.co.jp

はサービス名一つでも，Service 記述，Profile 記述，Process 記述，および Grounding 記述の間で引用しあうための整合性を保たなければならない．そのような OWL-S 記述の負担を軽減する目的で OWL-S エディタがあるが，OWL-S をコンパイル結果とするようなタスク言語にも同様な理由がある．

第2章では OWL-S 1.1 に新たに導入された仕様の問題点を指摘し，第3章でタスク言語設計の前提としてのエージェントについて述べ，第4章でタスク言語の設計方針について報告する．

2. OWL-S 1.1 の問題点

2.1 エージェントによるウェブサービス発見

ServiceProfile における Profile 記述には ServiceProcess で定義される IOPE が現れる．ServiceProfile におけるカテゴリー記述は，UDDI のようにエージェントが必要な情報を得るきっかけとしては有効であるが，エージェントにとってあるサービスが最終的に利用可能なものかどうかを決定するためには，ServiceProfie 記述および ServiceProcess 記述における IOPE のドメインオントロジー記述とエージェントの手持ちデータとの意味論的マッチングが必要である．ここで意味論的マッチングとは，エージェントが IOPE のパラメータクラスに関する豊富なドメインオントロジーを知っていることを前提に，素朴には IOPE のパラメータクラスに関する包摂概念によってサービスを同定することを言う．

OWL 記述による抽象的なサービス記述を現実のウェブサービスにグラウンディングするには AtomicProcess を WSDL の Operation に，Input と Output を WSDL の message に対応させる Grounding 情報が必要となる．現在，C# や Java でウェブサービスを構築すると，自動的にその WSDL 記述を得ることができる．したがって，ドメインオントロジーとタスクオントロジーを用いて，WSDL から自動的に Grounding 情報を生成するツールが有効である．そしてドメインオントロジーを参照しながら，半自動的に ServiceModel 記述や ProcessProfile 記述を行うことが望ましい．しかしある組織でアノテーションしたサービス情報と別の組織でアノテーションした情報を互いに流通可能にするためには，そのための上位(upper)オントロジーのプラットフォームが必要になると思われる．

ここで設計するタスク言語は，タスク実行時にエージェントのサービスインタフェース部を介して，現実のウェブサービスに結合されるものとする．

2.2 エージェントによるウェブサービス合成

エージェントがウェブサービスを合成するためには，複数のアトミックサービスの IOPE に関する意味論的マッチングにより，ユーザの要求するゴールを達成するような，サービス呼出しシーケンスを生成する機能が必要である．ウェブサービスの IOPE は古典的計画プログラムである STRIPS に出てくる用語と同等であり，我々は SWS のための予備的研究において[島田 2003]，STRIPS 風計画プログラムを開発し，ウェブサービスの合成と自動実行を行った．そしてそこで得た経験を元に，セマンティックウェブ用エージェントの持つべき基本的特徴について考察した[小出 2004]．

OWL-S においてウェブサービス合成にかかわるオントロジーは ServiceProfile 記述と ServiceProcess 記述であるが，サービスの IOPE に用いられる値の値域としてドメインオントロジーにおけるクラスを指定する．実行時にはエージェントは前提条件を世界状態に照らしてチェックし，入力にはインスタンスを与え，ゴールに指定されたクラスのインスタンスを得るが，計画時には IOPE のクラス情報から包摂関係を見て，接続可能なサービスのシーケンスを導く．計画初期状態のインスタンスデータを入力と前提条件に利用できれば，シーケンスはそこでグラウンディングされる．

我々のウェブサービス合成においては，すべての複合プロセスは対応する単純プロセスの定義を有するとして，ブラックボックス的に考え，複合プロセスの中身となるコンポーネントについては実行時以外には考えないものとする．実行時には複合プロセスのコンポーネントがエージェントにより展開され，解釈実行される．

(1) ウェブサービスの入出力

OWL-S 1.1 で，ウェブサービスの入力と出力はルール表現用言語 SWRL の変数 swrl:Variable とされた．タスクオントロジストはこのサブクラスである process:Input および process:Output のインスタンスとしてウェブサービス入出力を定義しなければならない．通常のプログラム言語と異なり，OWL-S における入出力変数は値の入る単なる器というわけではなく，それ自身ある構造を持つ OWL-S 上のエンティティ定義になっており，IOPE のインスタンスとは別に実際のウェブサービスの入出力値があって，その型と値の情報が入出力変数の process:parameterType プロパティ値と process:parameterValue プロパティ値に保持される．エージェントはウェブサービスを実際に呼び出すときには，process:Input の process:parameterValue に入力値を入れてウェブサービスを実行し，実行後は process:Output の process:parameterValue に入っている出力値を取り出さなければならない．実際に process:parameterValue とウェブサービス間の値の受け渡しは，ここではエージェントのサービスインタフェース部において，別途行われるものとしている．

IOPE の包摂概念による意味論的マッチングには，パラメータの型ではなく，パラメータ値の型すなわち process:parameterType プロパティの値が用いられなければならない．この仕様は，ドメインオントロジストが独立に構築したドメイン知識を，タスクオントロジストが引用する仕組みとして，ドメイン知識とタスク実行上で必要となる知識を分離するために効果的と思われる．

(2) 局所変数とスコープ

ウェブサービスの入出力変数も抽象ウェブサービスからアトミックサービスまでそれぞれのスコープを持つが，そのスコープはその抽象あるいは特殊の定義範囲であるというのは自然であり（すなわち上位抽象の入出力は下位特殊から参照可能），スタックや連想リストで容易に実装でき，とくにスコープ概念を明示的に持ち込む必要もない．ところが 1.1 において，前提条件や条件付き効果を記述するために論理式を用いることにして，そこに局所変数を導入したために，変数のスコープと値の束縛について明示的に記述する必要が生じた．OWL-S 1.1 では「Owl Rules で行われているように，変数をグローバルな名前つきインディビジュアルとしてモデル化するのは誤りである」と述べられているが，エージェントが OWL-S コンテンツからウェブサービスの意味を理解してサービスの発見と合成をしようとしたときに，URI で指定されるオントロジー記述ではなく，局所変数についてその意味やスコープを理解するということは，プログラムの意味を理解することとほぼ同義の困難な仕事のように思われる．我々は前提条件や条件付き効果の記述における変数は，プログラム上の局所変数ではなく，Web-PDDL と同様に型付けされたものであり，URI で指定されるものであるとする．

(3) ウェブサービスの前提条件

前提条件すなわち process:hasPrecondition プロパティの値は expr:Condition のインスタンスである．expr:Condition とは expr:Expression のサブクラスであり，1.1 では expr:Condition のサブクラスとして expr:KIF-Condition，expr:DRS-Condition，expr:SWRL-Condition の三つが定義されている．その各々のスロット構成と具体的な表現方法は異なっているが，RDF のシンタックス中にそれと異なるシンタックスをどのように組み込むかで 1.1 では苦労している．そのトリック的解決策として，それらの表現式は RDF シンタックス上では rdf:XMLLiteral として RDF パーザを素通りさせ，OWL-S 処理系でそれを受け取って処理させるという方法を提案している．この論理式中に変数が用いられるが，その変数の値に何をどのように束縛させるかその仕組みが問題となる．OWL-S における変数は構造体であるのに対して，論理式中の変数は一般的にはそうではない．すなわち同じ変数名であっても実装上は異なっており，後述の条件付き効果における変数束縛と同じ仕組みを想定しているとすると，この OWL-S 処理系は入力変数（process:hasInput の値）あるいは局所変数（process:hasLocal の値）からその値（process:parameterValue の値）を取り出して，論理式中の変数にセットしなければならない．

前提条件や条件付き効果の論理式を評価するのは OWL-S 処理系の役割で，その評価結果は当然真偽値を返すことが期待されていると思われるが，OWL-S 1.1 には論理式の返値のクラスに関しては何も定義されていない．後述するように，条件付き効果は論理式の体裁をしていても返値が期待されているのでない．したがって，好都合なことに OWL-S 1.1 定義に違反しないまま，真偽値以外の値を使用することができる．（プログラマは任意の表現をこの expr:Expression のサブクラスと定義することで，前提条件に使用することができる．）現に我々は，ロケット打上運用支援システムのデモンストレーションにおいて[1]，入力には用いられないが，ウェブサービスを選択するための機構として前提条件を用いた．たとえば，「運転状態に対応したプラント情報収集プロセス」のサブクラスに「運転状態正常に対応したプラント情報収集プロセス」と「運転状態異常に対応したプラント情報プロセス」があり，前者の前提条件には「プラント運転状態正常条件」を，後者の前提条件には「プラント運転状態異常条件」を process:hasPrecondition の制約として記述した．このような方法で，変数を使用せずに，包摂概念による意味論的マッチングだけで，ウェブサービス適応のための前提条件本来の機能を実現することができる．

(4) ウェブサービスの効果

現実のウェブサービスの返す値が OWL-S のプロセスの出力となるが，それだけではエージェントがウェブサービス実行によって起こる副作用を知る手段がない．そこで副作用をウェブサービスの効果として OWL-S 記述する．したがって多くの場合，その中には出力変数が現れるであろう．あるいは極端にはそのウェブサービスには直接関係ない変数が必要になるかも知れない．OWL-S 1.1 ではこの効果の表現は expr:Expression とされているが，掲載されている例は論理式である．効果はその評価結果の真偽が問題ではなく，式中に現れる変数の値の変化で，副作用が表現される．エージェントが効果を必要とするのは，ウェブサービス実行による副作用を他に知る手段がない場合であり，もし該当ウェブサービスが副作用に関するクエリを受け付けることができれば(Amazon において配送の進捗を問い合わせるように)，効果をエージェント内で計算する必要はない．ユーザの関心事であるような重要なウェブサービスの副作用については，一般的にクエリを受け付けるようなサービス設計がなされるべきであろう．重要な世界の変化は，世界に聞くべきである．したがって，ここでは効果としては，ごく簡単な手続きを expr:Expression のインスタンスとして定義し，サービス実行後に自動的に実行する機能を考えることにする．

(5) ウェブサービスの入出力データフロー

ウェブサービスの合成には，必然的に（あるウェブサービスの出力を別のウェブサービスの入力とする）入出力のデータフローが伴う．OWL-S 1.1 ではゴールから逆向きのデータフローの記述（consumer-pull と呼ぶ）を行う．Performance 記述は Process とこのデータフローの関係を束縛情報として含んでいる．しかし，OWL-S 1.1 の入出力変数は構造を持っているため，単純に文脈上のスコープだけでは議論が終わらない．少し長くなるが引用すると，

> 「fromProcess の値域が Perform であるという事実から，微妙な問題が生じる．・・・実際には fromProcess = CP の ValueOf である valueSource の Binding を引用することはできない．なぜならば，CP は Perform ではなくプロセスだからである．もしそれを考えれば，プロセスから得られる値を引用することには意味がない．なぜならプロセスが実行されるたびに異なる値となるからである．パラメータの『現在値』すなわち実際の Perform 中の値，を参照する表現が必要である」．

つまり，Binding の valueSource である ValueOf の fromProcess の値は，一見プロセスに見えるがそうではなく Perform であり，エージェントが束縛実行時に必要とするのは，この Perform の実行時における process プロパティ値にあるプロセスの出力である．1.1 ではそのために特別な変数，TheParentPerform を導入した．TheParentPerform のコメントには「A special-purpose variable, used to refer, at runtime, to the execution instance of the enclosing composite process definition」とある．すなわち，TheParentPerform はプロセス実行時におけるそのプロセスを参照するための変数であり，OWL-S 処理系はオブジェクト指向プログラミングにおける Self や This のように，この変数に動的にプロセスをバインドしておかなければならない．しかし正確に言えば，OWL-S における process:Paramter (process:Input，process:Output，process:Local，および process:ResultVar)はプログラム言語における純粋な変数ではなく，構造を持ち，実際の値は process:parameterValue にあるため，上記議論は OWL-S におけるすべての変数について言えることである。process:Input と process:Output のエクステントとスコープは一致しており，そのままでは実行制御があるプロセスを離れるときに，変数のスコープからはずれるとともにその変数も参照不可能になる．現在普通に用いられているプログラム言語は静的束縛変数が用いられ，そのため OWL-S のパラメータも静的束縛であるかのように見えるかもしれないが，解釈実行系においては動的に扱わなければならず，スコープをはずれても変数の値の参照を可能にするような機構が必要である．もし解釈実行ではなく，OWL-S プログラム記述がコンパイルされて実行されるとすると，テキスト上の静的変数が実行時に正しく値を持つようにコンパイル結果を出力しなければならず，コンパイル時の環境と実行時の環境を正しく区別してコンパイルするというコンパイル技術が要求される．

(6) コンパイル実行か解釈実行か

エージェントの動作原理としてコンパイル実行を前提とするか，解釈実行を前提とするかの問題には，より徹底した議論が必要

[1] ISWC2004（広島）にてデモ展示

である. 先の報告[小出 2004]において我々はこの問題について考察し, 開かれたウェブの予測不可能なウェブの世界を前提とすると, エージェントは状況依存プラニングエージェントでなければならず, 解釈実行型でなければならないことを述べた. この問題に関係して, スコーピングについて我々の見解を述べれば, 一般的には, たとえばあるプロセスを終了したからそのプロセスにスコープされた変数をすべて忘れてもよいということは言えない. それが可能な場合は, 再びその変数の値が利用されることはないという保証があってのことであるが, そのためにはエージェントは現在状態からゴールに至るすべての可能性を調べ尽くさなければならない. 動的に変化する不確定なウェブ世界を前提にすると, これはそもそも成り立たない. すなわち, 変数のスコーピング問題は適応的エージェントの存在とそぐわない. もちろん人間であれ, 動物であれ, 昆虫であれ, 各存在の知的レベルに応じた記憶の能力がある. 不確実なウェブ世界に対応する知的なウェブサービスエージェント実現のためには, プログラムレベルでのスコーピング問題とは別に, 記憶機構の研究が必要である.

ここで考えるタスク実行においては, 当面単純な解釈実行を前提とする. 実行効率向上のための部分コンパイルは将来の課題とする.

(7) OWL-S における計算記述

従来から SWS のパラダイムでは, 簡単な四則演算にもウェブサービスを呼ぶのかという疑問があったが, 束縛記述を可能にしたついでに, OWL-S 1.1 ではデータフロー記述の一部として関数記述のための process:valueFunction, 定数記述のための process:valueData, 変数記述のための process:valueType が導入された.

一見高度化されたように思われるプログラム計算機能であるが, スコーピング問題と同様に, その記述の意味をエージェントは理解できるかという問題を抱えることになる. 我々のタスク記述では, その複合プロセスに相当するタスクのボディにウェブサービス呼出し以外の計算手続きを含むことはできるが, エージェントはボディの中身を見る必要はなく, そのタスクの IOPE のみでサービスを同定できるものとする.

3. エージェント設計

3.1 エージェントアーキテクチュア

タスク記述言語設計の前提となる, エージェントのアーキテクチュアを Figure 1 に示す. プラナー部は各タスクの IOPE とドメインオントロジーを見て, 与えられたゴール達成のための計画を行う. 計画結果はメモリー部に置かれる. メモリー部には, タスクオントロジー, ドメインオントロジーとともに, 現実世界の反映である部分を持つ. 実行部は逐次的に実行プログラムをメモリー部から取り出すが, メモリー部は実行部からの要求があったとき, そのときの状態によって, 与えられた実行タスク(クラス)をインスタンス化して, 実行可能な手続きを実行部に渡す. 実行部は与えられた実行手続きを実行し, 結果をメモリー部に返し, 次の実行タスクをメモリー部に要求する.

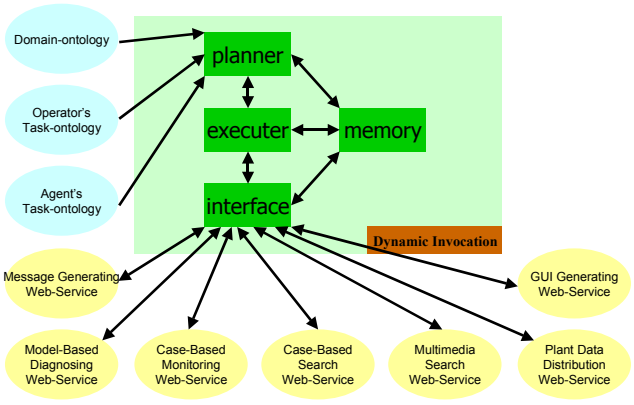

Figure 1 Agent Architecture

実行部がウェブサービス実行でタイムアウトになった場合や, 前提条件チェックで失敗した場合には, たとえば再プラニングなどが必要となるが, そのような各モジュールの全体的な統合された働きは状況依存エージェントの実装問題として解決されなければならない.

SIPE [Wilkins 1988] ではリソースの競合を解決するために, 監視と再計画と呼ぶ機能があり Plan Critics & Solvers というアーキテクチュアがある. 再計画機能と Critics 部でプラン結果の修正を行う. 事例ベース計画プログラム CHEF [Riesbeck 1989] のアーキテクチュアは Plan Retriever, Plan Modifier, Plan Storer, Plan Repairer, Assigner, Anticipator となっており, ゴールバリュー階層構造, 類似度測度, 修正ルール, Critics and Plan Specs, 失敗語彙, 修正戦略などの機能がある. 連続的計画実行フレームワークと呼ばれる CPEF [Myers 1999] は状況依存エージェントに最も近い実装と思われるが, プラナー (AP-SIPE による), プランサーバ (PRS), プランリペア (PRS), プランマネジャー (PRS), シミュレータ(PRS), インタフェース(PRS)からできている. ルールベースの認知メカニズムシステム Soar [Rosenbloom 1991] は短期記憶, 長期記憶を持ち, ゴール・サブゴールの階層構造, 選好, 問題空間によって推論が実行される. これらのアーキテクチュアを参考に, Figure 1 に示した計画部, メモリー部, 実行部を実装したのちに, 各モジュールと統合した状況依存の機能を開発する.

3.2 エージェントの記憶機構

SWS のための予備的研究において[島田 2003], 事例ベース推論のための記憶機構 MOP(Memory Organization Package) [Riesbeck 1989]を利用した. ただし, その時は RDF や OWL 記述ではなく, オントロジー記述も MOP で行った. ここではエージェントの記憶機構として, 事例ベース記憶機構を利用することにし, セマンティックプロセッサ SWCLOS[Koide 2004]をベースに MOP 風の機能を実装することにする. SWCLOS では, RDFS 語彙および OWL 語彙のすべてのエンティティは CLOS のオブジェクトである. すでに CLOS をベースに MOP を実装[Koide 2002]した経験があるため, SWCLOS をベースに MOP を実装することは我々にとって容易である.

事例ベース記憶機構 MOP には, スロットの束を与えてスロット値の包摂概念によってクラス階層構造の最も末端の正しい位置にそのスロット束を有するインスタンスを作成する機能がある. 記憶の中からスロット束にマッチしたインスタンスを取り出すときにも同じ機構が用いられる. この機能を用いれば, ある抽象的なタスクの下で, IOPE の具体的な値を与えて, その IOPE を適応可能な末端のタスクを探し出すことができる. 先の予備的研究[島田 2003]では, これを出力値がクラスでも動作するように拡張することで, ある入力データに対して適応可能なタスクを見つけ出し, 入出力値がクラスでも動作するように拡張することで, ゴールを達成するために必要なタスクを見つけ出した.

Srivastava と Koehler [Srivastava 2004] はセマンティックウェブサービスの記述は, ビジネスコンサルトの行うことと同様の性格のものでなければならないという動機から, 2ステージアプローチと呼ぶ手法を提案している. この方法ではオンデマンド・ワ

ークフロー合成部がスクラッチで抽象的なワークフローを合成し，ワークフローインスタンシエーション部が実行可能なワークフローを生成する．彼らの提案は我々と非常に近いが，前者がエージェント計画部，後者がメモリー部の機能に相当すると考える．彼らは抽象ワークフローと実行可能なプログラムの再利用性を主張するが，我々のシステムではそれは事例ベースの記憶機能によって自然に実現される．

3.3 抽象タスクの解釈実行系

実行部ではリスプ系の方言である Scheme をベースに，関数実行部を，抽象タスクをインスタンス化したのち実行するように拡張する．すなわち，エージェントは最初に抽象的な引数（クラス）を有する抽象的なタスク（クラス）の呼出し形式を読み，Scheme の eval の代わりに，メモリー部のインスタンスデータと入力引数（クラス）とのマッチングにより，その時点で適応可能な抽象的メソッド（クラス）を同定し，それをインスタンス化してから手続きを実行するものとする．

リスプの動作機構は read-eval-print ループであり，関数呼出し形式のリストの第1要素は関数であって，その実体は世界状態によって変化することはない．CLOS のメソッドでは，メソッド名だけでは実際の実行手続きは決定されず，メソッド名と引数クラスの組み合わせによって実際に適応されるメソッドが決定される．Scheme では，関数実体もリストの第1要素にあるシンボルを評価して得られるが，我々の抽象タスクの解釈実行系は，リスプや Scheme の read-eval-print に対して，read-instantiate-perform-print ループと言えるかもしれない．

これまで計画分野では解釈実行において PRS と RAP が用いられてきた．手続き的推論システム PRS [Myers 2001]には記憶部に相当するデータベース，ゴール集合，条件テストとエージェントの行動手続きをグラフ化したプラン集合があり，プラン中の要素が PRS によって解釈実行される．RAP (Reactive Action Package) [Firby 1995] のゴールとプランには様々な抽象レベルがあり，RAP システムにはセンサー記憶があって，表現言語によって表現された手続きを解釈実行する．

3.4 計画部

SWS のための予備的研究において[島田 2003]，STRIPS 風計画プログラムを開発し，ウェブサービスの合成と自動実行を行ったが，その経験から，無駄なウェブサービスの呼び出しや，ウェブサービス実行相互の干渉問題を解決するには，STRIPS のような古典的プラナーではなく，GRAPHPLAN に端を発する近代的プラナーでなければならないことがわかっている．これまで多くのプラナーが開発されているが，SIPE-2 の研究者らによって，実世界の複雑さとスケールの観点からいくつかのプラナーが批判されている[Wilkins 2001]．SIPE-2 は SHOP2 と並んで代表的な階層型タスクネットワーク計画(HTN)のプラナーであるが，実問題におけるユーザとのインタラクションを重視して，HTN は STRIPS 風のプラナーよりもユーザとのインタラクションを自然に実装でき，知識主導型のプラナーに応用しやすいと述べている．Disjunctive planning アプローチはユーザにとってのわかりにくさから否定されている．また，メリーランド大学の研究者が SHOP2 についてプラナーの完全性や健全性を主張することには，意味がないと述べている．状況依存のエージェントにおいてはプラナーの完全性はそもそも問題にすることに意味がない．

Russel らによる状況依存エージェントの計画部は UCPOP [Penberthy 1992]のような部分順序階層計画機能を有するプラナーが想定されている．UCPOP は様々な機能追加が行われて，現在に至っているが，これらのコードやその最新版である SGP[Weld 1998] のコードは公開されているので，状況依存エージェント計画部を作成することはこれらのコードを参考にして可能だと考える．

3.5 状況依存エージェント

先の報告[小出 2004]で，不確実なウェブサービスを駆動するエージェントは状況依存エージェントであると述べた．Russel は状況依存エージェントの擬似コードを掲載している[Russel 1995]．それを見れば，どのようにエージェントが設計されなければならないかがわかるが，残念ながら実際のコードはオンラインリポジトリにもない．Russel の記述に従って動作するセマンティックウェブ用エージェントを開発するには， RDF パーザ，RDFS 処理系，OWL 処理系，OWL-S 処理系，一階述語論理推論機能をインフラストラクチュアとして，知識ベースとのインタフェース（Tell と Ask）機能，クエリ機能，階層的プラン作成機能を開発し，因果リンクの管理，脅威や冗長の検出などを可能にする必要がある．

4. タスク言語設計方針

4.1 タスクの定義

ここで考えるタスクは OWL-S のシンプルプロセスやアトミックプロセスに相当するものを主要な側面として，Service 記述や Profile 記述を統合したものであるとする．必要であればタスク処理言語で記述された内容を OWL-S に変換するが，その場合には一つのタスク名から Process 記述におけるプロセス名，Service 記述における Service 名，Profile 記述における Profile 名が生成されるものとする．たとえば，タスク名が PlantInformationCollect であれば，PlantInformationCollectService，PlantInformationCollectProfile，PlantInformationCollectProcess が生成される．

タスクには OWL-S と同様に IOPE があり，入力と出力にはドメインにおけるクラス名が指定されるとする．前提条件と効果にもドメインにおけるクラス名を指定することができるものとする．

IOPE の記述のみのタスクは抽象タスクであるシンプルプロセスに相当するか，アトミックタスクに相当するものとする．すべてのタスクは階層的に組織化されるが，末端に位置するものがアトミックタスクに相当するものとする．

IOPE 記述のみでなく内部に手続き実体を持つものがコンプレックスプロセスに相当する．OWL-S の制御構造に相当する記述が手続きに許され，Scheme 風な制御構造表記を OWL-S に変換することができるものとする．この手続きの中には，任意のリスプ関数を用いることができるとする．通常の Scheme 関数と抽象的なタスクを識別する必要があるかも知れない．そのために，TLAMBDA という特殊なラムダ文を用いてもよい．また局所変数も導入してよいが，エージェントはタスクの意味を把握するためにボディの内容は見る必要がないとする．

タスク定義のために，CLOS の defmethod と同様なタスク定義用マクロ defTask を設計する．

4.2 関連研究

PDDL (Planning Domain Definition Language) [McDermott], は計画システムのためのコンペティション(AISPS)の公式タスク表現言語として設計されたものである．その後，時間表現，頻度表現など機能追加を重ねている．パンクしたタイアを修理するという，Russel による計画問題の PDDL による表現例（SGP より一部を転載）を示す．

```
(define (domain flat-tire-adl)
```

```
(:requirements :adl :universal-preconditions)
(:types container hub physobj - object
        nut tool wheel - physobj)
(:constants wrench pump jack - tool
            ground - object)
(:predicates (annoyed)
             (locked ?c - container)
             (open ?c - container)
             (in ?x - physobj ?c - container)
             (have ?x - physobj)
             (tight ?n - nut ?h - hub)
             (loose ?n - nut ?h - hub)
             (on-ground ?h - hub)
             (unfastened ?h - hub)
             (on ?n - (either nut wheel)
                 ?x - object)
             (free ?h - hub)
             (inflated ?w - wheel)
             (intact ?w - wheel)
             (jacked ?h - hub))

(:action loosen
    :parameters (?n - nut ?h - hub)
    :precondition (and (have wrench)
                       (tight ?n ?h)
                       (not (jacked ?h)))
    :effect (and (loose ?n ?h)
                 (not (tight ?n ?h))))
(:action tighten
    :parameters (?n - nut ?h - hub)
    :precondition (and (have wrench)
                       (loose ?n ?h)
                       (not (jacked ?h)))
    :effect (and (tight ?n ?h)
                 (not (loose ?n ?h))))
(:action jack-up
    :parameters (?h - hub)
    :precondition (and (not (jacked ?h))
                       (have jack))
    :effect (and (jacked ?h)
                 (forall
                   (?x - (either nut wheel))
                   (when (on ?x ?h)
                         (not (on ?x ground))))
                 (not (have jack))))

(:action inflate
    :parameters (?w - wheel)
    :precondition (and (have pump)
                       (not (inflated ?w))
                       (intact ?w))
    :effect (inflated ?w)))
```

前提条件と効果があり，変数に型があり，全称記号 forall がある．Dou と McDermott らはこの PDDL 表記と DAML 表記の間の翻訳を行う PDDAML を開発し，さらに異なるヘテロジニアスなオントロジーをマージするための Web-PDDL [McDermott 2002]を提案した．しかしそれらはいずれもウェブサービス実行のための処理言語ではない．我々のタスク処理言語は，本質的には OWL-S と同等のタスク表現言語であると同時に，直接エージェントがそれによりウェブサービスの発見，合成，実行を行うエージェント用の言語である．

5. おわりに

Figure 1 に示したウェブサービス用エージェントを前提に，エージェント設計について報告し，エージェントがウェブサービスの発見，合成，実行を行うためのタスク言語の開発を提案した．提案の動機は，ロケット打上運用支援システム開発における，我々の OWL-S の経験から，セマンティックウェブサービスの理念の正しさと，実際にオントロジーと処理系を開発する上での OWL-S の問題点を感じたからである．本報告の内容に従って実際に事例ベース記憶部，Scheme ベース実行部，部分順序計画部を開発し，動的に変化するプラント状態に動的に対応できるロケット打上運用支援用状況依存エージェントを開発する．

6. 謝辞

本報告は，文科省ITプログラム「ITを活用した大規模システムの運用支援システムの構築」の一部として実施されたものである．本プロジェクト実施では大須賀節雄東京大学名誉教授に技術評価委員長をお願いし，オントロジー構築に関して大阪大学溝口理一郎教授にご協力戴いている．記して感謝の意を表する．

参考文献

[Firby 1995] Firby, R. J., The RAP Language Manual, AI Lab, Univ. Chicago, 1995.

[Kiczales 1992] Kiczales, G., des Rivières, J., Bobrow, D.G., The Art of the Metaobject Protocol, MIT Press, 1992.

[小出 2004] 小出，島田，「セマンティック・ウェブサービス用エージェント」，SIG-SWO-A303.09, 2004.

[Koide 2002] Koide, S., MOP3: Memory Organization Package On Meta Object Protocol, ILC2002, 2002.

[Koide 2004] Koide, S., M. Kawamura, “SWCLOS: A Semantic Web Processor on Common Lisp Object System”, http://iswc2004.semanticweb.org/demos/32/, 2004.

[小出 2005] 小出，島田，「OWL-S 1.1 とエージェントによるウェブサービス実行 ― エージェントは OWL-S 1.1 を理解するか ―」，信学技報，KBSE2004-44，2005.

[McDermott] McDermott, D. V., The Formal Semantics of Processes in PDDL. Proc. ICAPS Workshop on PDDL.

[McDermott 2002] McDermott, D. and Dou, D., Representing Disjunction and Quantifiers in RDF, ISWC2002, pp.250-263., 2002.

[Myers 1999] Myers, K. L., CPEF A Continuous Planning and Execution Framework, AI Magazine, 20-4 (1999), 63-69.

[Myers 2001] Myers, K. L., Procedural Reasoning System User’s Guide, SRI, 2001. http://www.ai.sri.com/~prs/

[Paepcke 1993] Paepke, A. (ed.), Object-Oriented Programming – The CLOS Perspective, MIT Press, 1993.

[Penberthy 1992] Penberthy, J. S. and Weld, D., UCPOP: A Sound, Complete, Parital-Order Planner for ADL, Third International Conference on Knowledge Representation and Reasoning (KR-92), Cambridge, MA, October 1992.

[Rosenbloom 1991] Rosenbloom, P. S., Newell, A., and Laird, J. E., Toward the Knowledge Level in Soar: The Role of the Architecture in the Use of Knowledge, Architectures for Intelligence (ed., K. Vanlehn), LEA, 1991, 75-111.

[Russel 1995] Russel, S. J., P. Norvig, Artificial Intelligence A Modern Approach, Prentice-Hall, 1995.

[島田 2003] 島田ほか，「事例とモデルに基づくプラント運転支援，― ロケット打上を題材として ―」，第 46 回自動制御連合講演会，pp.511-514, 2003.

[Riesbeck 1989] Riesbeck, C. K. and Schank, R. C., Inside Case-based Reasoning, LEA, 1989.

[Steele 1990] Steele Jr., G. L., “Common Lisp The Language, Second Edition”, 1990, Digital Press.

[Srivastava 2004] Srivastava, B. and Koehler, J., Planning with Workflows – An Emerging Paradigm for Web Service Composition, ICAPS 2004, Workshop on Planning and Scheduling for Web and Grid Services, 2004.

[Weld 1998] Weld, D. S., Anderson, C. R., and Smith, D. E., Extending Graphplan to Handle Uncertainty & Sensing Action, AAAI-98, 897-904, 1998.

[Wilkins 1988] Wilkins, D. E., Practical Planning, Morgan Kaufmann, 1988.

[Wilkins 2001] Wilkins, D. E. and desJardins, M., A Call for Knowledge-based Planning, AI Magazine, 22-1, pp.99-115, spring 2001.