# Planning Process Instances with Web Services

Charles Petrie

Stanford University, Stanford, CA, USA petrie@stanford.edu
http://www-cdr.stanford.edu/~petrie
Presented at ICEIS AT4WS 2009:
http://www.iceis.org/Workshops/at4ws/at4ws2009-cfp.htm

**Abstract.** Planning is an important approach to developing complex applications composed of web services, based upon semantic annotations of these services. Despite numerous publications in recent years, the problems considered in the literature typically do not require planning as it has been well-defined in computer science. This could lead to confusion about which technologies are being designated, and raises the question of what whether planning is an appropriate technology for services.
We describe the essential features of planning technology and note its advantages, which include the dynamic synthesis of processes and the lack of need to verify the correctness of the message exchange.
We show that planning technology really is necessary by giving an example of web service composition that cannot be solved with simpler technologies as could previously published examples.
We describe the basics of adapting planning to web service composition. We restrict its use to process instance synthesis in order to simplify exploration of some fundamental issues. A major issue is that web services are usually incompletely modeled. We illustrate this with a second example. We show some additional semantic annotations of web services can be used to solve the problems similar to the example when used in conjunction with re-planning.

## 1 Introduction

We begin by defining "web services" as a general technology of remote procedures invoked with common Internet protocols conforming to descriptions of at least inputs and outputs in some machine-readable Internet generic syntax and accessible with common Internet protocols[3]. This definition captures the distinctive properties of this class of technology and the salient properties needed for planning: a description of the service that can reasoned about prior to service execution. Such a general definition also avoids being tied to particular protocols and syntax that are popular among industrial software developers at any given time but includes them.

These input and output descriptions can be thought of as semantic annotations, when they are sufficiently declarative, similar to the information required for planning. It is necessary in general to further add preconditions and effects to service descriptions to apply planning technology. With such semantic annotations, planning has the potential to synthesize automatically processes composed of web services using well-known techniques.

## 2 Process Instance Synthesis

The general problem of synthesizing programs is very hard and so if one would like to construct arbitrary applications using web services, that problem will be almost as hard even if web services have the advantage of being components with restricted inputs and outputs. Thus we would like to further restrict the kinds of these applications in certain ways. Often this is done by saying the output must be a web service itself, or a workflow in some particular format, such as WS-BPEL[1]. We start with an different but strong restriction.

In general, a planning problem always has a goal state of the world expressible in some form of logic. Our restriction is that we will only consider service planning problems in which all of the variables in such a goal are instantiated. Some of the discussion may indeed apply to goals with free variables, but we only require logical propositions for the goals of the planning discussed in this paper.

The plans that result from composing web services together constitute a process. Such a process consists of calling a designated set web services in a certain order with particular inputs. When the goal has fully instantiated variables, we will call the resulting plan a *process instance*.

One reason for this restriction is that classical planning only deals with such fully instantiated goals. In the so-called "blocks world", we say that we want to achieve a state in which blocks *A*, *B*, and *C* are stacked in a certain order. The science and technology of planning in this completely modeled world is well-developed and understood. We would like to apply such planning directly to planning web services and can almost do so, if we have instantiated goals. We do so by declaring the services to be planning operators, corresponding to those in the blocks world such as *(STACK ?X ?Y)*.

Another reason is pedagogical. Some technologies work to develop algorithms for automatically synthesizing workflows. We restrict planning to automatically synthesizing a process that would be a single instance of such a workflow. The power is still the same and the issues of planning can be more easily explored.

The utility of this restricted approach is still significant. Given a goal state, some information about the constraints governing the process based upon business logic, and the current state of the world, including databases, planning can synthesize a single-use process that achieves the goal if such a process is possible. The result is the same as if the workflow were re-generated each time in response to changed conditions, except that the problem is simpler. The resulting process, as we shall see, is guaranteed to be correct.

## 3 Planning

For general purposes, it will suffice to say a planning technology has the capability to reach some goal state $G_f$ from some initial state $G_o$ by ordering instances of performance of some subset of actions $\{A_i\}$. Each action may change the state of the world: it has *effects*. Conditions that were previously true may no longer be true in the state

[1] Business Process Execution Language for Web Services: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

that exists after the action is performed. Such effects may affect the *preconditions* of other actions in the plan: conditions necessary for the action to be correctly performed.

In a correct plan, the the preconditions of each action in the order are satisfied in the state in which the action is called. There are a number of formalizations of planning in the literature published over the last thirty years, at least consistent with our informal characterization. The situation calculus[1] is a variant of this general description that avoids defining a state but is otherwise consistent with the description above.

Some planners perform planning by deductive synthesis, the most elegant version being situational calculus. The sequence of actions is constructed as a proof of the goal state. When so proven, a correct plan is also *sound* in the logical sense. This has the advantage that no further properties about the the messages exchanged among the services need be verified: the message exchange implicitly defined by the plan is necessarily deadlock-free and will terminate.

There may be service composition approaches that do not construct a plan by proof. In order to avoid specifying the technology used for planning, we characterize a planner as a technology that given a planning problem, can produce a plan that could be formalized as a proof of the goal.

However a plan consisting of the action sequence $\{A_p\} = \{A_0, ...A_i, ...A_n\}$ for goal $G_f$ is synthesized, it is a correct plan when $\forall(A_i, S_i)$, where $S_i$ denotes the $ith$ state, in which $A_i$ will be called, $preconditions(A_i, S_i) \wedge effects(A_i, S_{i+1})$ where $S_{i+1}$ is the successor state in which $A_{i+1}$ will be called; and $G_0 \wedge \{A_p\} \models G_f$.

But then do we require a planner to be complete? There are few planners that can solve any planning problem for which there is a solution. But if we do not require a planner to be complete, then what is to keep us from claiming a particular technology is a planner when it simply reproduces a previously programmed solution to a single problem when given it? And there are different classes of planning problems. Research needs to be done to define such classes and say that a planner is a planner for that class if it can solve all of the problems in that class, or perhaps, less restrictive, at least an infinite number of them. But we ignore such issues for now and concentrate on the idea of creating a correct plan.

## 4 Simple Web Service Planning

Now let us proceed to describe a simple web service planner by first stating some fundamental assumptions.

A plan consists of a partial order of web service calls, which results in a partial a set of states ordered by the relation *subsequent*: $>$. Let $(p = a, S)$ denote that input or output property $p$ has value $a$ in state $S$. It would be more elegant to make these special cases of preconditions and effects, but we have found it efficient to treat them differently.

Let service $W_S$ be called in state $S_i$ with some inputs and preconditions , and return outputs and effects in state $S_f$. Then $S_i < S_f$; $\forall\ p, (input\ W_S\ p),\ \exists a \mid (p = a, S_i)$; and $\forall p, (Output\ W_S\ p\ b), (p = b, S_f)$.
This $b$ is not the unknown value at execution time but rather a skolemization of that value.

Further, there is no *spoiling*: we have the persistence frame assumptions: $(p = a, S_k) \Leftarrow (p = a, S_i) \wedge (S_i < S_k) \wedge \neg\exists S_j, (p = b, S_j) \mid S_i < S_j \leq S_k \wedge a \neq b$.

We may have *constraints* that are service-specific or global conditions that restrict property values and service calls in state sequences, and *preferences* that are property values and service calls in some state sequences which are preferred.

For a really simple planner, we would like to have the meta-constraint of single-valued properties: $\neg((p = a,\ S) \wedge (p = b,\ S))$. However, the alert reader will see that such a seemingly innocuous assumption will cause a problem with our simple formulation: you can't use the property names twice in successive states.

Suppose the plan is to withdraw money from Johnny's account in state $S_1$. The `Balance` in $S_1$ is, say 100. The next step in the plan to deposit money in Frankie's account. What is the input value for `Balance` in $S_2$?

The mistake in the formalization above was in conflating the web service part names with the actual object property. Our formulation should be $(p.o = a, S)$, where $o$ is the object of which $a$ is the value of the property $p$ in state $S$. We shall have to have separate objects that represent Johnny's And Frankie's accounts in our plan representation.

With that refinement, we can now talk about *conditions*. These will be described by *fluents*, which are relations upon features of the state of the world. In $On(A,\ B)$, $On$ is a fluent.

Since our inputs and outputs correspond to the blocks world, a condition will consist of a fluent and a set of such properties: $\{(p.o = a)\}$. If we want such a condition to hold in state $S_i$, we will say $fluent(\{(p.o = a)\}, S_i)$. The preconditions and effects of web services are such conditions. We can make inputs and outputs a kind of condition as well by using the distinguished fluent $know$ to say that the condition is that we know the values of these properties in a state.

We can now give a *simple* pseudo-algorithm for composing web services to achieve a goal.

**PlanSingleGoalCondition**($S_0, S_F, G,$):
**P1.** Given Goal of some single condition $G = F_g(\{(p.o = a)\}, S_f)$, a set of preferences and constraints over the solution that are fixed, and conditions $\{C_i\}$ true in initial state $S_o$, call procedure
**FindGoal**($S_0, S_F, G,$):
**F1.** Attempt to prove that $G$ is already true in the target state $S_f$, where $S_0 \leq S_f$.
**F1.1** If true, return $S_f$.
**F1.2** If no proof is possible, find the set of web services $\{W_S\}$ to be called in state $S_i$ with an effect, or output (if $F_g = know$), $E_O$ such that $G$ unifies with $E_O$ for each $W_S$ in the set.
**F2.** Select one, $W_S$, from among the equivalence set of services that conforms constraints of the problem, selecting first those that conform also to the problem preferences. Do not use any previously selected in the next step. If no more exist, fail.
**F3.** Attempt to unify the inputs, outputs, preconditions, effects of $W_S$ with $G$.
**F3.1** If one will not unify,or if a constraint is violated, fail with $W_S$ and select another in **F2.**
**F3.2** If successful with $W_S$, Mark the effects and outputs as being true in state $S_f$ and the inputs and preconditions as being true in state $S_i$, that $S_i < S_f$, and that $W_S$ was

called in $S_i$. Return $S_i$.
**End of FindGoal**
**P2.** Let $\{Conditions\}$ be the set consisting of each precondition $G'$ of $W_s$ as currently instantiated, and, for each input of $W_S$, $I = ?value$, perhaps with variable $?value$ bound by unification so far, add to $\{Conditions\}$ $G' = know(I = ?value, S_i)$. Then order the set $\{Conditions\}$ so that the Input conditions are selected first in the next steps.
**P2.1** Select the first $G' \in \{Conditions\}$ and let $S_r$ = **FindGoal**($S_0, S_i, G'\}$)).
**P2.2** If the last call to **FindGoal** failed, fail.
**P3.** If $S_r$ results in new states and unifications, propagate new instantiations of variables in the inputs to the preconditions, outputs, and effects of $W_S$ in states $S_i$ and $S_f$. Repeat **P2** with the rest of $\{Conditions\}$ until all are done.
**P4.** Return the call of $S_f$ and all other states and calls as marked.
**End of PlanSingleGoalCondition**

The only reason that inputs and outputs are distinguished conditions and that inputs are attempted to be achieved before preconditions is simply that it is often efficient to do so, since preconditions may have arguments consisting of several of the inputs.

This simple procedure suffices for many cases of web service composition problems. However, this is not yet planning because it does not deal with conjunctive goals in which multiple conditions should be achieved in the goal state, which is what makes planning hard.

## 5 A Simple Web Service Planning Challenge Problem

A web service planner should to be able to handle many web service planning problems with the kind of action effect/precondition interference required to solve the famous Sussman anomaly[8], which is caused by conjunctive goals, the individual plans of which interfere with one another.

The simple procedure defined above is a constructive one. One way to make it a planning algorithm would be to modify it so that given a conflict among conjunctive goals, it would either make a new subplan or re-sequence the current action sequences. This would make it a goal-regression[7] planner. There are other ways to solve this problem. Any procedure that completely explores the space of all possible action sequences will also solve conjunctive goals though this may be inefficient.

Being able to solve conjunctive goal problems like the Sussman anomaly is a fundamental test. If it can't, it's not a planner. At least this test for planning gets us beyond analogous US Supreme Court method of recognizing pornography that "we'll know it when we see it."[2]

Many, if not most, web service planning problems in the literature do not seem to require planning with conjunctive goals, or even preconditions and effects other than the inputs and outputs of web services. For instance, [5] only considers the inputs and outputs of web services together with the fluent "know" as preconditions and effects, but these inputs and outputs can never interfere with each other. Planning is not required.

[2] Jacobellis v. Ohio, 378 U.S. 184, 197 (1964)

We contribute here the outline of a web service problem that requires planning, possibly unique in the web service planning literature in its requirement to handle action effect/precondition interference.

Consider the following supply chain problem for car manufacturer *CENTRA* which provides a CD player with a CD dispenser in its cars. Supplier WeCDs offers a CD player that it has already connected to a power supply. Supplier CARCDs offers a CD dispenser. All of these systems use a unique connection technology that allows both power and signal to flow among the daisy-chained components, which may be connected in any order. The physical connections are unique to the company that makes the connections, and can only be disconnected by the company that created the connection. Any company that makes a connection offers a service to disconnect that connection.

Supplier *UNIBUS* offers a service for connecting CD components that are shipped to it with instructions, if any, for the order of components. *CENTRA* needs the CD Player connected to a CD dispenser connected to a power supply because of how the components will fit to the car chassis. *CENTRA* has a service that can connect one component to the car chassis. Each CD component has two (2) ports for connections: the chassis has unlimited ports.

A shipping service can move components (connected components are also components) from company to company. What is the shortest plan that meets *CENTRA*'s requirement, ignoring the niceties of purchase orders and requirements, and considering only the services that offer, connect, and ship the components among the companies? (Actual formulation of the services with their effects and preconditions is left as an exercise to the reader.)

What happens in this problem is that constructive planners will try to ship the component consisting of the CD player and power supply from WeCDs and the component CD dispenser from CARCDs both to UNIBUS with instructions from CENTRA to connect the CD player to the dispenser and the dispenser to the power supply. Then the connected components are shipped to CENTRA. There is no port left to connect to the chassis. The subplan to fix this is quite long and re-sequencing should result in the shortest plan.

## 6 The Travel Expense Approval Problem

We now give an simpler web service problem in more detail that does not have conjunctive goals but which illustrates a fundamental problem with planning web services.

It is sometimes pointed out that web services may create new objects, unlike the blocks world which is closed, but this is not the main difficulty. The related fundamental problem for planning is that web services are "black boxes". I.e., they may only be incompletely modeled. We know always the exact result of *(STACK A B)* is *(ON A B)* in the blocks world. We are not assured of the analogous result for web services.

At plan time, we may not know the value of an output of a planned service. The simplest example is a request for a stock quote. We can plan to know it and expect that there will be a value when the service is executed. But at plan time, often the best that we can do is to skolemize such an output by assigning a unique name to its value.

The meaning will be that *there exists* a value at plan time. (In the Appendix, we use a naming convention.)

As an example, consider a corporation that has the following policies in place for processing travel expense claims:

- Travel Clerk can approve under $5K
- Request to manager needed otherwise
- Unless the requester is a manager
- But requesters can't approve themselves
- Who is a manager is determined by HR Policy
- Currently requires 10 direct reports
- Seniority currently based upon number of direct reports
- Use the least senior manager to authorize if necessary.

Further, suppose there are the following facts about the company:

- $NumberDirectReports(Melanie\ Ralston\ 10)$
- $NumberDirectReports(Jackie\ Brown\ 12)$
- $Department(Max\ Cherry\ Travel)$
- $Role(Max\ Cherry\ Clerk)$
- $Can-Authorize(?authority\ ?requester) \Leftarrow$
  - $Requester-Name(?claim\ ?requester)$
  - $Requested-Amount(?claim\ ?amount)$
  - $(<?amount\ 5000)$
  - $Department(?authorityTravel)$
  - $Role(?authority\ Clerk)$
  - $(\neq ?authority\ ?requester)$
- $Role(?manager\ DepartmentManager) \Leftarrow$
  - $NumberDirectReports(?manager\ ?num)$
  - $(>?num\ 9)$

There are some services available to accomplish the reimbursement of a travel claim request described in the appendix (with some omitted parentheses and formalizations to be discussed).

A minimal process instance that would satisfy a request by Ordell Robbie to reimburse a claim for $5000, meeting all of the company polices, is illustrated by figure 1 in which we go from a state in which certain key facts are true to another state in which new facts are true because we have called a web service. Each fact that is the output or effect of one web service satisfies the input or precondition of the next service, except for the first fact that Jackie Brown is qualified to approve the request, which is simply provable and the basic employee information for Ordell Robbie (not shown above) that is needed for the service that gets the employee bank information. Also, note that this is a partially ordered set of actions, rather than a sequence, as this last service can be called anytime prior to calling the employee reimbursement service.

Had the goal been to reimburse Ordell Robbie for $4000, then the process would have been the same except that the service requesting authorization would not have been called as it would have been provable that Max Cherry could have approved the request. This is the difference between writing a workflow and writing an algorithm that produces a correct process when needed: the workflow would have included conditional flows for all cases, and the process instance only works (possibly) for one case.

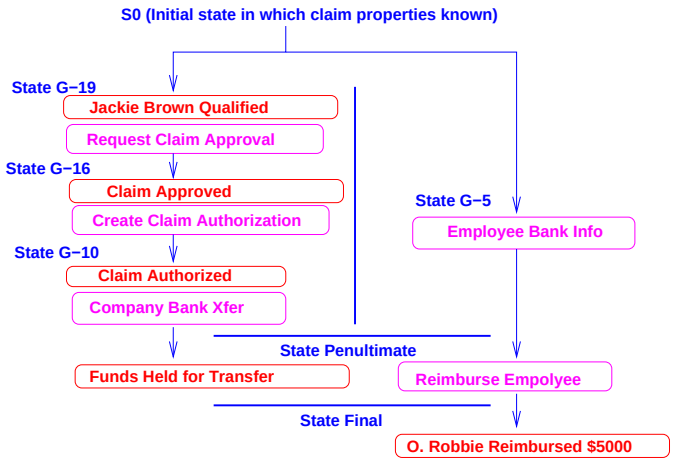


**Fig. 1.** A Process Instance Solution

## 7 Using Default Values

Jackie Brown was used because she was the least senior manager. In order to respect such policy preferences, a straight-forward use of many technologies, such as theorem proving, will require that all solutions be produced and then the preferred solution tried first. This is less efficient than proving which path should be next pursued in a depth-first exploration of the search space, as in the simple procedure previously shown.

What if Jackie Brown declines to approve the request? Then then next least senior manager should be tried. This illustrates the fundamental problem of incomplete modeling for planning. We don't know which, if any, of the managers will approve the request.

At least traditional synthetic deductive approaches require that plans cannot fail because they must be sound. And as we have seen, the very definition of planning requires some idea of sound proof, no matter what technology is used.

One approach is to make conditional plans that decide on plan branches based upon sensing. But in the travel expense problem, the next manager, Melanie Ralston, may also decline to approve the request. There is no way to make a plan that is guaranteed to succeed somehow, even with conditional planning.

Pragmatic planning of web services (at least) needs a new theory of soundness. One candidate might be analogous to paraconsistency entailment[2] where we identify all conditions that could cause failure and prove that the plan is sound when these are excluded.

Without waiting upon a new theory of soundness, we can pragmatically make plans that are very simple process instances because we can simply re-plan when one of these failure conditions occurs at execution time. If Jackie Brown refuses authorization, we repair the plan with a request to Melanie Ralston. Such interleaved execution plan repair seems a promising pragmatic approach. This also has implications for the semantic annotation of web services.

But it is not enough to say that the output of the service `Request-Claim-Authorization` is of type `Authorization-Request`: we must be able to say what the possible output values are. An annotation such as `Possible-output-values Request-Claim-Authorization Authorization-Request (Approved Denied Deferred))` is required. We may also make the stated value of `Authorization-Request` the keyword *RETURN* to indicate the possibility of selecting one value. Such an annotation may be derivable from the existing web service description. But we need a further annotation that must be derived by knowledge engineering.

In order to know how to plan with the possible output values, we need to know that approval is a reasonable default with which to plan, so we need a semantic annotation such as `Selectable (Request-Claim-Authorization Authorization-Request Approved)`. The planning practice of choosing a likely default such as approval, because it satisfies an plan condition, might be called *Hopeful Thinking*.

We need such an annotation because we wish to avoid *Wishful Thinking* in planning, in which there is no reason to choose one of the possible outputs except that it works for goal we wish to achieve. A good example might be the plan to buy a new car, which only requires that we we buy a winning lottery ticket. Such plan is as at least as likely to fail as not.

Another semantic annotation required for general web service planning is whether or not the web service always returns the output value for given input values whether or not the service is a mathematical function. Consider a web service that gives your drivers license number given your name and date of birth, versus one that gives you a stock quote given the ticker symbol. The latter is a function only when the time of execution is included as an input value, which may, though not always, present a problem for planning. This is a special case that requires additional service annotation.

This example does not show this requirement, nor the requirement to handle loops, which is known to be problematic. For instance, consider the web service that in response to a single input, outputs serial values, concluding with one that means *Finished*. At least this kind of loop can be inefficiently handled simply by making the exit condition the desired output of a service and then repairing the plan with the same subplan until that output is returned at execution time[3]. This also requires additional service annotation to inform the planner that it is not being insane by trying the same subplan repeatedly and expecting a different result. These are all additional challenges for service planning research.

## 8 Conclusions

We define planning and show the basics of adopting planning to the solving of composing web services into a process that achieves a fully instantiated goal. Apart from problematic considerations of completeness, a program is a *planner* if the resulting goal state is entailed by the plan and it can solve conjunctive goal problems. Web service examples that require conjunctive goal solving are rare and we provide one.

[3] Suggested by Georg Jung, U.Potsdam.

The definition of planning that we give allows us to distinguish between the technology used to build a planner and planning in the sense of entailment. Even Java[4] can be used to write a planner. The seminal paper [4] about composing web services describes a method of programming the goals, preferences, and constraints of the problem with Golog: the resulting formulation is turned over to an interpreter that functions as a planner.

We describe the basic requirements of producing a plan that is a process instance, which is simpler than producing a workflow. Because of the possibility that some web services at execution time will return outputs that conflict with the current plan, or simply fail, sound planning is inadequate for planning web services.

If some outputs (and by extension effects) can be assumed by default, semantic annotations of these, in addition to preconditions and effects, can be used in conjunction with re-planning to solve problems of synthesizing process instances from web services. This kind of defeasible reasoning will require a revision of sound entailment for planning.

We have used Redux[6] as a Goal-Operator-Oriented Programming (GOOP) method to build a planner that can achieve the examples in this paper by declarative expression of preferences as operators and goals to try first, constraints that must not be violated, characterization of the undesired output values as *contingencies*, and by using the resequencing method for conjunctive goal interference.

Finally, the reader is encouraged to try the two very simple examples in this paper on their own service planning technology. The example of the car manufacturer outsourcing the CD player may be the first example in the web services planning literature to require real planning.

**Acknowledgment:** This paper was sponsored by SAP Labs USA and benefited from discussions with many people, especially Michael Genesereth,Tim Hinrichs, Sheila McIlraith, Daniel Meyer, Harald Meyer, and Richard Waldinger.

## References

1. Finzi et al.: “Open world planning in the situation calculus”, it Proc. AAAI 2000, AAAI Press, 2000. Available at http://logic.stanford.edu/serviceplanning/readinglist/openworldsitcalc.pdf
2. Kassoff and Genesereth: “PrediCalc: a logical spreadsheet management system”, *The Knowledge Engineering Review*, **22**, Cambridge University Press, Nov 2007, pp 281-295.
3. Ludwig et al.: “Cross Cutting Concerns”, *Dagstuhl Seminar 05462 on Service-Oriented Computing*, November 2005. Available at http://tinyurl/webservdef
4. McIlraith and Son: “Adapting Golog for Composition of Web Services”, *8th Int. Conf. on Knowledge Representation and Reasoning (KR2002)*, Morgan Kaufman. , April 2002
5. Oh,Lee, and Kumara: “A comparative illustration of AI planning-based web services composition”, *ACM SIGecom Exchanges* , **5**:5, pp 1-10, ACM, 2006.
6. Petrie: “The Redux’ Server”, *Proc. Internat. Conf. on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, May, 1993.
7. Pollock: “The logical foundations of goal-regression planning in autonomous agents”, *AI Journal*, 106, 1998, pp 267-335.
8. Sussman: “A Computer Model of Skill Aquisition”, American Elsevier,

[4] http://java.com/en

## Appendix: Web Services for Travel Expense Approval

- Web-Service Create-Claim-Authorization
  - Input Create-Claim-Authorization Requester-Name
  - Precondition Create-Claim-Authorization Authorized" ( (Requester-Name ?requester) (Authorization-Request Approved) (Authority ?authority))
  - Output Create-Claim-Authorization Claim-Authorization RETURN-Create-Authorization

- Web-Service Request-Claim-Authorization
  - Input Request-Claim-Authorization Requester-Name
  - Precondition Request-Claim-Authorization Qualified-Authority (Requester-Name ?name Authority ?authority)
  - Effect Request-Claim-Authorization Authorized (Requester-Name ?requester Authorization-Request ?approval Authority ?authority)
  - Output Request-Claim-Authorization Authorization-Request RETURN-Request-Claim-Authorization

- Web-Service Company-Bank
  - Input Company-Bank Requested-Amount
  - Input Company-Bank Currency
  - Input Company-Bank Claim-Authorization
  - Effect Company-Bank Held-for-Transfer (Requested-Amount ?amount Currency ?cur )

- Web-Service Employee-Bank
  - Input Employee-Bank Requester-Name
  - Output Employee-Bank Employee-Bank-Number RETURN-Employee-Bank-Number
  - Output Employee-Bank Employee-Bank-Name RETURN-Employee-Bank-Name

- Web-Service Reimburse-Employee
  - Input Reimburse-Employee Requester-Name Employee-Bank-Name Employee-Bank-Number Requester-Address Requester-Name Requested-Amount Currency
  - Output Reimburse-Employee Confirmation RETURN-Reimburse-Employee-Confirm
  - Effect Reimburse-Employee (Reimbursed Requester-Name ?name Requested-Amount ?amt)

And there is a company banking policy expressed as:

- Precondition ?service Held-for-Transfer (Requested-Amount ?amount Currency ?currency) $\Leftarrow$
  - Effect ?service Reimbursed (Requester-Name ?name Requested-Amount ?amt)