

Article

Enterprise Coordination on the Internet

Charles Petrie

Computer Science Department, Stanford University, Stanford, CA 94305, USA;

E-Mail: petrie@stanford.edu; Tel.: +1-650-269-1516; Fax: +1-650-725-7411

Received: 11 November 2010 / Accepted: 11 January 2011 / Published: 17 February 2011

Abstract: Enterprises are now connected internally and externally to other Enterprises via the Internet in ways that are increasingly difficult to manage, especially as these interconnections become more dynamic. Current methods of coordinating the effects of change as they propagate through these networks of connections are not likely to scale. What is needed is a new paradigm for how the Internet supports such coordination. Indeed, the Internet should and could provide fundamental coordination functions that are missing today. In this paper, we describe how such a “Coordinated Internet” would work (this paper is an expanded version of [1]). The key functionality of a Coordinated Internet would be that the Internet actively watches what people do (analogous to search completion on desktops today), correlates these activities, and actively notifies people when and how their current tasks affect and are affected by the activities of other people. This would be accomplished by standard coordination functions implemented as a common Internet layer that can be used as a utility by more specialized applications. Such a Coordinated Internet would revolutionize enterprise management, for all enterprises, large and small, corporate and personal. For example, static workflows would become obsolete for all but the the most routine processes. Some solutions provide existence proofs of such a coordination substrate, such as the Redux solution in concurrent engineering, which we describe herein. However, foundational research remains to be done in the new field of Coordination Engineering in order to reach the goal of a future Internet in which coordination functions are fundamental.

Keywords: internet; enterprise; coordination

1. From Web 2.0 to Semantic Web Services

The key distinction of the ill-defined Web 2.0 technologies from older ones is that in Web 2.0, publishing data is more democratic: anyone can publish with no programming. Such Internet (not just web) technologies have contributed to the rise of “Emergent Collectives” [2,3] that often cause unexpected disruptions in the extant economic models.

Web services offer a next step in this process of dynamic Internet technologies. To be precise, by “web services”, we do not just mean WSDL [4] but any technology that is more than just a Remote Procedure Call, because it has a published description, according to the Dagstuhl definition [5], such as SA-REST [6] but not REST by itself.

The effect of the distinction of web services having public descriptions is that, in principle, users can consume public services without having detailed knowledge of how the service is implemented. The descriptions should also be “open” in that they are more standardized than, say, the descriptions of a library of code subroutines. This intention is often described as “semantic”, more about which in the next section.

Web services should be also democratic, which in this case means the service need not be changed by the user. Democracy has an additional sense in that now programs can play as well as humans. Web services could also be “just-in-time”, which means the user may invoke the service at any time without prior arrangement.

As a consequence, we could have a free market of services: users have a runtime choice among competing services. Especially important for enterprises, this provides flexibility: one could find an alternative service whenever a contingency blocked use of the default service.

Were the democratic and just-in-time principles to apply for all public services on the Internet, they would comprise an open free market of just-in-time services, but development of dynamic open services seems far in the future for various practical reasons [7].

2. Service Semantics

Even a limited web service democracy depends critically upon standardized descriptions of services, typically envisioned as the Semantic Web [8]. By “services” in the rest of this paper, we mean them as defined in [9], which include but are not limited to web services. Services may be implemented in a variety of ways, including mobile applications.

It is beyond the scope and intention of this paper to discuss “semantics” and we have previously done so in [10] and [11]. Briefly, we equate semantics with interoperability: the more of one, the more of the other. Were everyone everywhere to adopt the same standards for computationally representing everything, semantics would be complete. In practice, technologies for providing semantics for different purposes and communities are various.

There are strong requirements for semantics in services, depending upon how they are used. We assume in this paper that the semantics problem will get solved one way or the other, perhaps by linked data [12] for example. We now discuss briefly some of the possible ways this may occur, even though it is not the main point of this paper, because it is so fundamental.

If one wants to dynamically and automatically compose services, then the semantics and the planning technologies must be very sophisticated. The best way to do this is with various sorts of logics. Unfortunately, only academics will understand sentences such as $F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee (F(s, s) \wedge \neg\gamma_F^-(\vec{x}, a, s))$ [13] and academics tend not to strongly influence industry very quickly. The three most likely scenarios for the development of service semantics are as follows.

The first is the development of “industrial service parks” [14]. There is a strong need for semantics in these closed applications because of the need for re-usability and service composition within so-called “Service-Oriented Architectures”. These semantics are likely to be more along the lines of standardized parameters for subroutines, rather than logics. The problem with this is that logics address problems that are likely to occur but be ignored in the rush to product.

For example, ignoring the failure of UDDI [15], industrial service parks are likely to index services by categories, rather than their specific effects, because the former is easier to do. They are not likely to provide sophisticated service composition because they want to control the processes at a low-level. Rather, the service descriptions are likely to be sufficient only for pre-defined precedence relationships among services, eliminating much of the rationale for services as a distinct technology with new functionality. They are even less likely to address issues such as default assumptions (for approval of requests for example), contingencies, and dynamic re-planning.

It may be that customers of the large providers would demand improvement but this has not been historically the case. So, we can expect poor semantics that will become more sophisticated over the years, as problems are exposed.

Consumer applications on the web and mobile devices are likely to drive simple semantics for vertical markets. Examples are social networks and travel-based sites that offer semantics for the attributes of people and for uniquely identifying geographic locations. RDF [16] databases are increasingly used and major engines are offering increasingly semantic search functionality [17].

As these applications become more interoperable, there will have to be pragmatic solutions to distributed updating and different namespaces. As with the industrial parks, these kind of semantics may not address some problems. However, this approach has the advantage that some consumer-oriented language for expressing simple business service descriptions, such as that the hours of operation are 9 am to 5 pm PST, may develop whereas they are not likely to in the less-democratic service parks.

A related approach that is likely to show good results is the development of new applications, such as “SEAmail” [18]. Such applications would provide adequate semantics for specific applications by initial design. This would typically be done by entrepreneurs and we decline to speculate further about these new applications except for the possibility of a “World Wide Workflow” as an example of a new workflow model made possible by semantics as a first example of what was posited in [8] about workflows and coordination that semantics would make possible.

We first examine the problem of static workflows that would be overcome by the dynamic workflows of a World Wide Workflow. We will then note that this is but an intermediate step towards a more radical change in enterprise management that is both possible and necessary.

3. The Universe Does Not Run on Workflow

Enterprises today are run largely by sets of static processes, which we will loosely call static “workflows”, although this term may have a more narrow definition in some contexts. The essential point is that these are pre-defined procedures authored by a (workflow) programmer who has had to identify company policies (such as managers need to authorize requests over a certain amount) and who has had to foresee all contingencies that should be handled in the future.

In contrast to this, events happen in the world because objects obey the laws of physics rather than programming. The ballistic arc of a thrown ball is not prescribed by a written procedure, but rather is a result of the laws of gravity and momentum. Even a guided missile takes into account these laws and makes physical adjustments to arrive on target. These laws can be made explicit and computations can take them into account. Were the universe run like enterprises, these laws would be implicit in code and a path computed for each ball to be thrown in anticipated directions.

Running an enterprise by such static procedures has the effects that managers never know whether their corporate policies have been correctly represented and workflow users are frequently trapped inside an overly-complex process given what they want to accomplish and their current context. Such static workflows also cannot easily respond to changes in policies and to contingencies.

There are supply chain problems, resulting from a needed change in suppliers, that could not be handled by today’s workflows, such as the example below, explicated in [19].

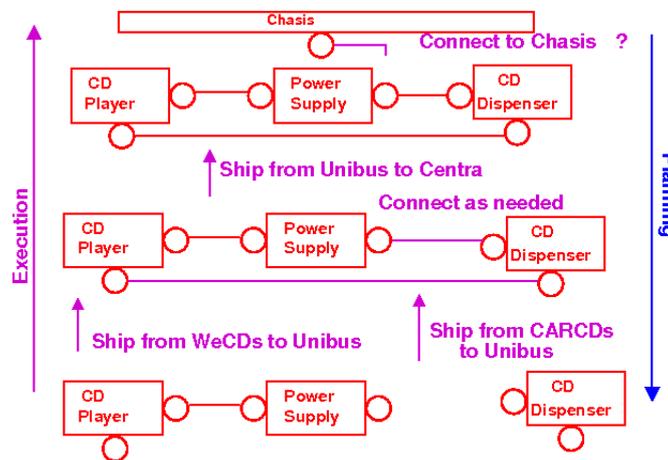
- CENTRA is a car company that needs a CD player connected to a CD dispenser connected to a power supply.
- The technology used is a special power/control connection that can only be disconnected by the company who made the connection.
- Any company that makes a connection offers a disconnection service.
- Each component has 2 ports for connections.
- UNIBUS offers a service for connecting CD components.
- CENTRA has a service that can connect any CD component to the car chassis via a free port.
- A shipping service can move components (connected components are also components) from company to company.
- CARCDs offers a CD Dispenser (advertised in a service).
- CDsr’Us offers a CD Player and a power supply (advertised in a service).

The workflow, a sequence of service calls, for handling this standard supply chain is obvious. The components are shipped to Unibus that makes the connections according to instructions from Centra, and then ships the finished macro-component to Centra, which connects it to the chassis.

But now there is a contingency: CDsr’Us cannot deliver the power supply in time. A search of services, based upon their semantic descriptions, shows that another company, WeCDs, offers a CD player connected to a power supply, which can be delivered in the needed quantities on-time. Let us ignore for now the management of change and focus on the question of how the workflow should be changed to take advantage of this solution to the contingency.

A naive approach to workflow modification would be to simply substitute WeCDs for CDsR’US as a provider of the CD player and power supply components. Figure 1 illustrates the subtle flaw in this approach. The “planning” arrow refers to the construction of the workflow.

Figure 1. Workflow Modification.



A naive substitution of CD suppliers results in CENTRA not being able to connect the final product to the car chassis because all of the connectors are already used. Standard workflow techniques will not detect this problem until the workflow is executed. The expensive fix to this requires shipping the whole system back to WeCD’s with a request to disconnect the CD player and power supply, and then shipping back the result.

Recognizing that the world is committed to workflow technology, we now outline one way that semantics would be used for a dynamic workflow system that could overcome the problems of static workflow systems.

4. A World Wide Workflow

An intermediate, incomplete solution to static workflows would be a dynamic workflow, open to everyone, scoped within some domain, and in the extreme, open to everyone on the Internet. The degree of scoping is determined by agreement upon semantics. We now sketch out what one possibility for this would look like as an expository device, rather than making a technical proposal.

Given the agreed upon semantics, there could be a Process-XML (PXML) that would describe workflow tasks, as well as a protocol for exchanging PXML messages, along with process servers. Each task description would have a control section and a work description. The task description declares the previous task step in the workflow, the next step, and the performers of both. The prior step represents history and the next step represents the plan yet to be executed. The control part of the PXML would include standard workflow control constructs. These include to whom notifications should be sent if an assigned task has not been completed by certain times as well as forks and merges.

The work section describes the input and output documents required for the task. Again, the input documents described are provided and are a part of the execution history. The output documents are planned and constitute the current criterion for task completion. “Document” in this context may be very abstract and even consist of a statement that the task has been completed. In addition, there may

be pre-conditions and post-conditions associated with the task. The pre-conditions should have already been met when the task is assigned. The task actor should fulfill the post-conditions.

Tasks are assigned to actors at first in the planned workflow, which may be incomplete. The workflow is monitored by a central server. Anyone within the domain of work, which might be the whole open Internet, may receive a task assignment. This might be the result of an open auction bidding on tasks, or from a search of capable actors.

Upon receiving a task assignment, the actor may refuse or accept it. The actor may also change the workflow by modifying not only the output documents and post-conditions, but also the control part of the PXML describing the task. The only part that may not be changed is that part describing the execution history so far. Aside from this, since the PXML is an editable description, both work and control sections for planned work may be changed.

Upon completing the task, the new PXML is sent to the next task assigner(s). However, before they receive it, the process server, which is mediating these communications, may object to changes because they violate certain policy constraints or because no viable workflow can be re-computed based upon the changes. The extent to which individual actors may also impose constraints, for instance making some inputs/outputs mandatory, requires further design analysis.

Such an open workflow system could be available to anyone on the Internet. The requirements would be (1) agreed-upon message and task semantics, (2) client software able to receive, edit, and send task messages, (3) communicating with the central server (of which there may be multiple ones for multiple purposes). This software could be web-based, email-based, or app-based.

The vision of “flash companies” described in [3] would be enabled and adequate semantics would likely quickly emerge for vertical markets as the need arose to better describe tasks, in auctions for example.

The important point here is to see that the next step in workflow technology is dynamic workflows. And once one sees this, it is only a short step to making workflows obsolete entirely.

The initial process template used for a open World Wide Workflow can be formulated by advanced technologies for composing services, since tasks and services will have much the same semantics.

More important, some form of dynamic process synthesis and re-planning is needed to implement the workflow reformulation caused by task actor changes. The alert reader will note that a World Wide Workflow protocol does not by itself solve the supply chain problem of Section 3, even if we have sufficient semantics for interoperable dynamic workflows. The dynamic process synthesis and re-planning technology required is full AI planning, which involves the formulas of the sort mentioned in Section 2, though such formulae can be hidden from the user. This leads to a more radical vision.

5. Enterprise Physics

Many technologies for the dynamic composition of web services have been developed in the last 10 years. If we can discover and consume reusable services with common descriptions, new processes may be dynamically constructed automatically to achieve state-based goals [19] upon demand. This would allow the workflow re-formulation needed for the World Wide Workflow. However, we then no longer need workflow at all.

Given a goal, e.g., to be reimbursed for travel expense, a new process can be created for this purpose, even just one time for one person and one set of expenses. The processes would take into account all of the currently expressed enterprise policies, e.g., what amounts must be approved by managers and the conditions that qualify to be a manager.

Individuals inside companies, and in cross-company projects, could synthesize workflows and other processes *as needed*. Companies could manage such processes by stating explicit policies and constraints that such processes would respect, instead of trusting that programmers would properly interpret these policies and waiting for new policies to be statically encoded. We call this *Enterprise Physics*.

Such dynamic processes should and could allow for change: contingencies, conflicts, opportunities, policy revisions, and outright failure.

This would indeed be a closed world in the sense that any model must be closed. It should also be noted that in some cases, the model will be deliberately scoped to work only within an intranet. Nevertheless, it would be better than the current model represented by static workflows in fixed code. As soon as a policy changes and is so stated, all process synthesis would take it into account immediately and explicitly. The same holds true for any contingencies: as soon as they are stated, the model and thus the execution changes, because in Enterprise Physics, the model is what is executed.

There are various ways that enterprise physics could be accomplished. The simplest was suggested by [20] in which relational databases and rules express corporate policies explicitly. An initial advantage is that they only have to be written once and are automatically used in all use cases. Furthermore, the kind of technology used is increasingly familiar to corporate IT.

Less familiar are various kinds of formal computational logics, ranging from datalog to full first-order logic. These allow almost any kind of policy to be expressed and used in computation. A major advantage of such logics is that the answer to queries can provide very sophisticated information based upon all possible considerations, quickly and automatically.

Some new work in differential logic provides ways to effectively compute workflows [21]. The “laws” of the enterprise can be written and edited as desired by authorized managers, and then users can ask the system for answers and processes that are guaranteed to be correct and respect all policies and constraints because they are essentially logical proofs.

Services provide an important approach to dynamic processes, provided they have an adequate semantic description, such as the tasks of the World Wide Workflow. The AI planning approach to automatic composition of services has been well established by a number of papers in the last decade. There are essentially two approaches: automatic generation of static workflows and dynamic generation of single-use process instances upon demand. We advocate the latter and explain methods in [19].

Full deductive (*i.e.*, logical proof-based) synthesis of processes leads to processes that are guaranteed to be correct and require no verification. When full planning is used (versus simple backward chaining), then conjunctive goals, required for the problem of Section 3, can be handled.

Finally, an added benefit is that the user knows that the process has been custom-built for the current problem or goal and does indeed solve it. There is no way of knowing exactly what static workflows actually achieved.

This is especially important with respect to process merging. When this is done with static processes, for example when two companies need to combine their procedures to work together, the final process

may be verified to see that it is safe and deadlock-free, but there is no proof that it meets any particular goal or end-state. In contrast, deductive synthesis produces processes that require no verification and provably accomplish some particular end state.

Though it would require a major revolution in enterprise IT, various relational database and formal logics, together with services and AI planning, provide demonstrable solutions to enterprise physics: running on explicit enterprise laws that must be obeyed and constructing correct processes to achieve goals as needed.

One (and there are several) objection to this technical vision is that it does not scale. However, it is more likely that the solutions to scaling for logic-based technologies will be developed, than that current approaches will be scaled to meet the highly complex and dynamic world of distributed global supply chains, especially as more and more use is made of services over Internet and mobile-based communications.

This (finally) raises the question of how global dynamic processes could be managed, the main topic of this paper. If we have global dynamic processes with distributed actors in various companies operating asynchronously, responding to contingencies and opportunities, the result could be nearly uncontrolled chaos, which is why we currently have static workflows.

6. The Coordinated Internet

Should we achieve the vision of dynamic processes, there is an even more radical future Internet that is possible and even necessary: the “Coordinated Internet”. This is a vision of a *pro-active* Internet that not only facilitates sharing and collaboration but also actively coordinates humans as well as various programs, most notably services, which are being consumed and provided in distributed dynamic processes. This is in contrast to mere collaboration, in which a system only supports human-initiated sharing of information.

One definition of coordination is dependency management [22], which means that changes within some system are propagated according to some model. More specifically, we mean by “coordination” the notification, typically asynchronously, to the right people of changes within a design, especially a process. Process components depend upon one another’s features and when these features change, sometimes the process should change but the right users should always be notified of the effects of the change propagation.

One example of a collaboration mechanism that is not coordination is Google docs [23]: this manages the internal dependencies to keep the documents consistent but does little to coordinate the users. This is a system that invites people to share but does not otherwise help them in distributed work: authors must invite one another to share the documents.

An example of a limited coordination mechanism would be a collaborative version of the logical spreadsheets [24] such that when one user modifies a field on his display, appropriate updates take place in related fields in other spreadsheets on the Internet. When such a system detects a conflict (e.g., a constraint violation) for which there is no defined automatic solution, it must somehow notify all of the users involved and prototypes do this by changing the colors of affected fields. This is a minimal example of a system notifying someone that they have been affected by change and that they need to take action.

A Coordinated Internet would go far beyond this: it would actively watch what people do (analogous to search completion on desktops today), correlate these activities, and actively notify people when and how their current tasks affect and are affected by the activities of other people. Example notifications are that a conflict has occurred and who is involved, safe and consistent solution options upon request, thrashing warnings, opportunities for synergy, and that some tasks are no longer necessary.

The trick is to do this in a useful way, so that coordination is increased, but people are not annoyed by a “big brother paper clip”. We can see what the minimal requirements are by looking at a distributed supply chain of the future.

7. A Future Supply Chain Scenario

In the future, car manufacturers will not make cars, they will only manage processes. In the hypothetical DellAuto’s supply chain, chassis parts are made in Shanghai, then assembled by ChassisMax in Mexico City, and then shipped to EnginesR’US in Stuttgart. Now suppose there is a new opportunity: Tasmanian-based AssemNow can do the next twenty chassis cheaper and faster. What is required to revise DellAuto’s supply chain on the fly?

First, there must be a check that the ChassisMax contractual penalty for canceling the current order is less than the savings of using AssemNow. If this is so, then the following notifications need to be made to:

- ChassisMax and AssemNow canceling and placing order.
- DellAuto’s Finance Department saying that the deposit with Bank of Mexico is no longer required.
- DellAuto’s Treasurer saying that the scheduled payment to ChassisMax should be canceled, and replaced by a penalty payment.
- DellAuto’s Order Department saying that a PO should be issued to AssemNow.

EnginesRUs should not be notified because message exchange with the shipping service shows that the chassis will arrive in Stuttgart on the original date. A coordination system would note only effects of the changes, as they propagate through the dependencies, and not make useless notifications just because something changed.

The effects of such functionality could be world changing, revolutionizing not only how companies are managed, but how any large enterprise is done, especially ad hoc ones, such as global relief efforts for catastrophes [25].

As noted previously, there are supply chain scenarios that cannot be handled by workflow and new technology is needed. It would be difficult to handle even the example of DellAuto’s supply chain with the World Wide Workflow without some coordination mechanism.

There are various ways to handle the management of dependencies and changes that propagate through them. The logical spreadsheet mechanism of [26] does so by noting whether or not “false” can be deduced from any of the logical expressions and then denoting which values of arguments to predicates lead to these inconsistencies. This requires paraconsistency: being able to reason sensibly even when there are logical inconsistencies, which is useful since logical inconsistencies may arise from a number of sources in distributed systems [27].

Another general technology proposed is that of [28] in which dependencies among web services are represented. Like the logical spreadsheet, this is a generalized notion of dependencies. Neither propose a specific method of notifications, although both provide the basic mechanism for identifying who is the owner of the variable value assignments that are in conflict.

We summarize in the next section a solution that does elaborate coordination notifications, which has long existed in the field of concurrent engineering [29], and is a model consisting of special dependencies for general design covering many situations. This model specifically defines a set of notifications that should be sent in specific situations.

8. One Mechanism for Coordination: Redux'

The Coordinated Internet can be viewed as the management of a configuration or planning problem worked on by distributed actors. When there are multiple objectives and no single objective function, as there is in the case of large projects with many engineers, the best we can do in terms of optimality is to ensure that no single objective solution can be improved without harming the solution of another: *Pareto optimality*.

The Redux model of design, as implemented in a subset called Redux' [30], implements the use of Pareto optimality within a model of design [31]. The full Redux model is a planning-inspired programming model in which operators are selected to apply to a goal and such a selection is called a "decision", the result of which must be at least one of a new goal or an assignment. The last may be the assignment of a value to a variable but may in general be any statement about the emerging design.

The general method of full Redux may be thought of as goal/operator-oriented programming. A subset of the system, Redux', manages the dependencies among all of the Redux objects, each of which may be valid or invalid depending upon changes and their defined propagation.

Full Redux can be used to implement full AI planning and process synthesis of the kind required for Enterprise Physics. Redux' is the underlying coordination engine, which can be used with almost any planning and reasoning system sufficient for Enterprise Physics. Redux' implements a specialized dependency management model that is nevertheless generally applicable.

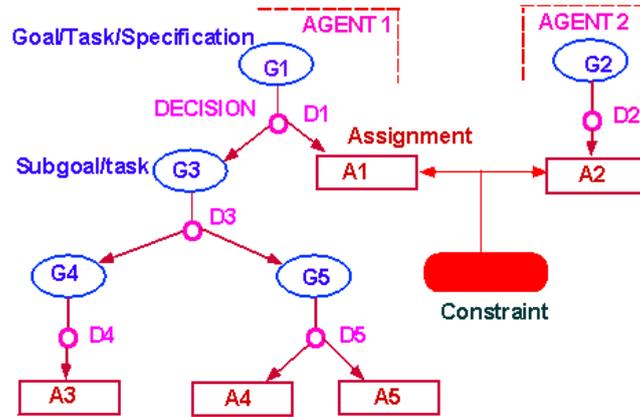
Logical spreadsheets and most general approaches to dependency management only consider this degenerate case in which every decision is exactly which value to assign to a single-valued variable, making the Redux model an overkill. However, many cases of coordination require much more elaborate goals and decisions.

Given a conflict (either a constraint violation or a goal block) and a solution (the revision of some design decision), using a justification-based truth maintenance system (JTMS), Redux' constructs justifications for revisions resulting from the resolution of conflicts using Dependency-directed backtracking (DDB) that become invalid if the conflict would not obtain if ever there is another way of resolving the conflict, thus enforcing Pareto optimality. This justification should not introduce any unsatisfiable circularity into the JTMS network.

Figure 2 illustrates the Redux' model with a simple conflict. Two actors, Agent 1 and Agent 2, have each tried to satisfy their individual goals with decisions D1 and D2 respectively. These have resulted in assignments A1 and A2. When both are valid, they result in a constraint violation. In this instance, since no other decisions support these assignments, the resolution of this conflict will require retracting either

D1 or D2. Neither agent can retract the other’s decision. Both will receive a notification of the conflict, and both can request a list of possible solutions.

Figure 2. Redux’ Example Conflict.



Were A1 to retract D1, Redux’ would cause the goal G3 to become invalid. This would not invalidate decision D3, but it would invalidate its optimality. The owner of D3 would then receive a notification to the effect that D3 is no longer needed and may be retracted. Should the owner of D3 decide to do so, then G4 and G5 would become invalid and the owners of decision D4 and D5 would receive similar notifications.

A1 will also be notified that although G1 was previously satisfied, because all of its subgoals were satisfied, this is no longer the case. Should A1 make some new decision to satisfy G1, this new decision would receive a justification for its optimality based upon the choice to resolve the previous conflict by retracting D1. Should either the constraint underlying the constraint violation or decision D2 lose their validity (for various reasons), this justification of optimality for the new decision would then become invalid and A1 would be notified that there is now an opportunity to return to the original (and presumably initially preferred) decision without causing the original conflict.

Notable characteristics of the Redux model include (1) distinguishing between goals that need to be achieved, and constraints that should not be violated; (2) distinguishing between conditions that affect the optimality of a design decision and its validity; (3) identifying opportunities, resulting from loss of Pareto optimality, as well as conflicts.

Distinguishing between goals that need to be achieved and constraints that should not be violated has proved generally useful in configuration technology [32]. Achieving a goal or resolving a constraint violation are two types of tasks managed by Redux’.

Logical spreadsheets (as well as constraint satisfaction systems) usually represent everything as constraints but implicitly have goals when a constraint resolution cannot be automatically determined and users must choose. Some versions of logical spreadsheets have also incompleteness, for instance, when there is a “oneof” type of constraint, which is a kind of goal.

One current commercial system [33] also explicitly represents such necessary variable choices. Redux extends this notion to general goals that must somehow be satisfied, as it is an abstraction of AI

planning in which some state should be achieved while respecting certain (typically temporal sequence) constraints.

The distinction between optimality and validity is important for managing the propagation of change. In Redux, the optimality of a decision depends upon the validity of the goal to which it is applied as well as various reasons, which may be environmental facts.

In full Redux, these optimality reasons were used in reasoning about the best choice of an operator to apply to a goal, but they may be any reasons a decision maker wants to state. A typical reason for choosing an airline flight, for example, might be its cost. Were that cost to change (prior to buying the ticket), the user should be notified that the reasons for choosing this flight have changed. However, the plan should not be changed automatically, as it would have to be were some contingency to occur, e.g., the cancellation of the flight. (Full Redux includes a notion of preference as it allows local choosing of operators according to explicitly-stated binary preferences and recording the reasoning used to do so, which may be full first order logic, as well as reasoning about the best fix for a conflict. This functionality can be used in Enterprise Physics to establish company policies, although it cannot be used to determine global optimality, which is in any case often impractical.)

In general, it is not a good idea to invalidate a decision simply because of loss of optimality, as this will cause changes to propagate through the distributed design and it is not always worthwhile to do so. For instance, just because there is now a flight that is two Euros cheaper than the flight chosen may not be worth changing extensive travel plans with several flight connections.

A special case of loss of validity of an optimality reason is Pareto optimality loss. If a user responds to a conflict by retracting a decision and making a second decision for the same goal, Redux' assumes the first decision was a local optimum and constructs a special justification supporting the validity of this reason for the optimality of the second decision. This justification will become invalid should changes make the initial conflict no longer necessary.

For instance, if another decision owner retracts (for some reason) a third decision that conflicted with the first, which is now retracted, then that first decision could be made again, indicating a loss of Pareto optimality. Alternatively, since constraints in Redux' may also have conditional validity, if the initial constraint loses validity, this would also be a loss of Pareto optimality. In all cases, the decision owner is notified of the opportunity to remake the original decision.

A set of notifications useful for active coordination is provided in Redux' and formally defined in [30]. The simplest example is when a subtask has become redundant because the method of achievement of the supertask has changed as seen in the previous example.

An example set of notifications built-in to the Redux model are as follows:

- You have been assigned a goal
- Your goal is no longer valid because a higher-level decision is invalid
- Your decision should be reconsidered because
 - A reason has changed
 - An earlier decision no longer conflicts
- Your goal is no longer satisfied because
 - An input to your decision is invalid
 - A subgoal is no longer satisfied

- You contribute to a constraint violation/goal block
- Here are the known safe and consistent solution options you requested.

In some cases, there is no way to achieve a goal without causing a constraint conflict. A user may declare this as a goal block. Goal block conflicts are treated nearly the same as constraint violations. All of the constraint violations blocking the different ways of achieving the goal are accumulated by DDB and the set of decisions involved is used to send notifications to the decision owners. At least one must retract their decision in order to resolve the goal block.

Redux also guarantees no thrashing among changes by decision owners leading to cycles and identifies safe solutions to conflicts and goal blocks in a distributed environment by management of decision rationales and the justifications produced by conflict resolution.

In the case of both constraint violations and goal blocks, the minimal subset of an AND/OR tree of decisions that could be retracted in order to resolve the conflict is generated dynamically. The solutions are safe in that they will introduce no unsatisfiable circularities in the network and they are minimal in the sense of subsumption. This is done in Redux' using a version of DDB that dynamically computes the minimal AND/OR tree of decisions that could be retracted in order to resolve the constraint violation. Though such a minimal subset of choices is usually computed in an Assumption-based Truth Maintenance System (ATMS) for all possible worlds, it is computed dynamically as needed also in the ATMS-based constraint handling system of [33].

9. Implications

Redux' notifications may be formalized messages exchanged among agents, services, or general applications. ProcessLink [34] was a prototype concurrent engineering system using Redux as a coordination manager. This was an agent-based system with a set of well-defined messages to be exchanged among the various agents.

Figure 3 shows a typical ProcessLink architecture. It is important to note that the coordination of a design among various designers can be extended to plans to build the design, as a plan is also a design. In [29], we described how a design system may be linked by these notifications to a construction planning system so that when the unfinished design changes, the schedule can be automatically optimized in response to such change.

In Figure 4, the displays (which were Java apps) for the designer are on the left and the displays for the construction scheduler are on the right. The upper two screens show the initial design of a Pick-Up Head (PUH) and the schedule to build it. Initially, the PUH was to be a custom design, so that the weight of the PUH was unknown until after it was built. Thus building the mechanical support for the PUH had to be scheduled after the PUH was built, because the support had to be of the correct strength.

In the lower left screen, the designer has changed the decision to build a custom PUH and is instead using a commercial off-the-shelf PUH. The old design is colored red to indicate it is now invalid. The assignments and one subgoal that depended on this old decision are now also colored red. One of the new assignments of the new decision (green) is that we know the weight of the PUH because of its specifications.

Figure 3. A ProcessLink Architecture.

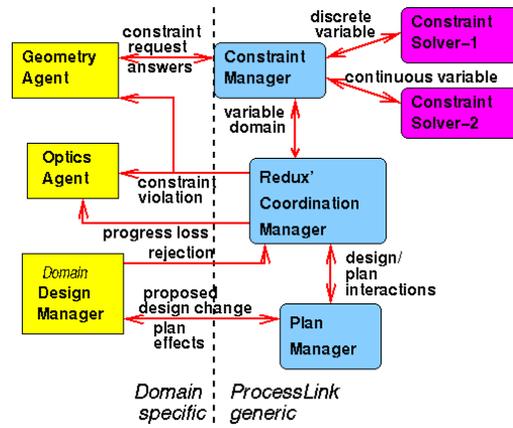
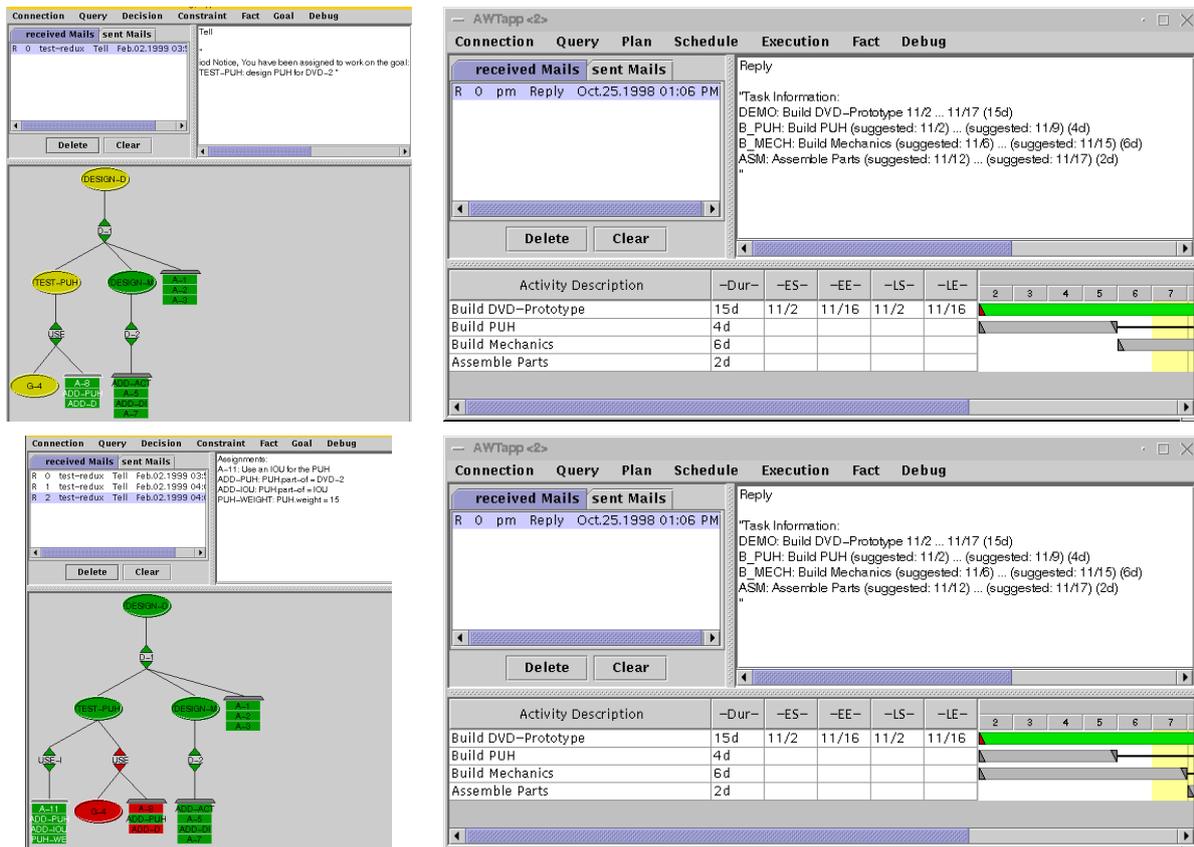


Figure 4. Design/Schedule Coordination.



The design system notifies the scheduling system of this change. The scheduling system, also using Redux', sees that the scheduling of the mechanical support only after the construction of the PUH is now a suboptimal decision and can revise it. If the scheduler is programmed to do so, this can be an automatic revision in the schedule that makes the whole schedule shorter.

Obviously such functionality allows fast-track designs in which construction may start before the design is complete. But in general, the ability to coordinate the architect, the general contractor, the inspectors, and the various sub-contractors is badly needed and could significantly lessen the cost and time required to design and construct any large project.

This clearly extends to just-in-time supply chains, especially if semantic services allow the discovery of better services and products, and the dynamic synthesis of processes to construct supply chains as needed.

This extends to any large human endeavor, such as coordination of relief for catastrophes, and also to small endeavors such as the formation of a band and design for booking and traveling to venues. It is easy to see that specialized “wizard” applications would be developed for vertical domains to speed the construction of such plans and managing the execution and revision of them.

Enterprise coordination is the next leap forward for the interoperable Internet.

10. Coordination Engineering

Redux’ is an existence proof of the kind of coordination dependency management and notification model needed for the Coordinated Internet. It is sufficient to manage the scenarios described in this paper, though there is much work to be done, as described in this section.

There are other coordination models besides Redux’ as well as outstanding issues not solved by Redux’. The features of Redux’ could be implemented as extensions to both [27] and [28].

The paraconsistency approach may be necessary in any case because logical inconsistencies are certain to arise in any distributed complex system. However, it is worth noting that both Redux’ and the system described in [33] are based on versions of truth maintenance systems, in which the nodes are propositions and are not reasoned about other than to detect constraint violations, which are arbitrary, not logical.

However, full Redux does use logical inference to reason about choice of operators, which goal to work on next, and a variety of conditions. Though the justified nodes are not part of this inference, underlying facts are and so may need paraconsistency, although Redux’ is agnostic about the inferencing method and even non-deductive methods could be used as proposed in [35].

In general, there is a need for much more research in the field of *coordination engineering*. Topics include the proper technologies for detecting and understanding tasks and filtering notifications so that they are more useful—instead of distracting or disturbing—to people.

Known research issues include (1) understanding how to combine constraint satisfaction techniques with DDB [36], (2) better algorithms for underlying justification graph control, (3) better representations of committed actions and sunk costs, and (4) providing appropriate transparency of information in supply chains. The last one is an especially challenging research issue. The end supplier in the chain needs to know about schedule and product changes by suppliers far down the chain. It is clear that such information can be passed up the chain as needed but it is not clear how to keep other suppliers from seeing this information, which may be necessary because of competitive conflicts.

Another general computer science issue is conflict resolution. Redux offers no specific advice as to which solution might best resolve a conflict. There is a long history of both domain-dependent preferences and domain-independent advice (for example, when there is only one choice, that decision could be retracted automatically.) There is some relation to on-going work in what to do when databases are inconsistent, though most of this research has to do with providing minimally consistent answer subsets. There is no good general answer to this issue now.

Coordination should also ideally provide advice for users to consider changes in decisions. This could be done in three levels. The first would be passive mode: consider propagating the effects of the proposed decision change and report the calculated consequences. The second would be an active “ping”: notify the owners of all other decision owners and consider their responses. The third level is what Redux actually does now: make the change and work with others on resolving the resulting conflicts, if any. How to best implement such functionality is a good engineering issue. Providing guidance using sensitivity analysis is a particularly interesting scientific question for both decision revision and conflict resolution.

Detailed questions about implementing coordination in the Internet involve what part of the coordination functionality should embed in what Internet layers. For instance, could the message notification be handled by smart routers and distributed servers? How should the “watching” function be implemented in browser plug-ins and mobile device apps? In Next-Link [31], the precursor of ProcessLink, we hand-inserted code in standard engineering programs. Is there a more scalable approach?

The Coordinated Internet also has deep issues in common with Internet of Things, such as authorization to change descriptions of products and services.

The Coordinated Internet provides a rich new topic of research for computer science as well as the potential to radically improve mankind’s ability to manage complex projects. We have enough technology to begin to build it.

References and Notes

1. Petrie, C. The Future of the Internet is Coordination. In *Proceedings of FES-2010: Future Enterprise Systems Workshop 2010*, Berlin, Germany, 20 September 2010.
2. Petrie, C. Emergent Collectives for Work and Play. Available online: <http://www-cdr.stanford.edu/~petrie/revue/> (accessed on 13 January 2011).
3. Petrie, C. Plenty of Room Outside the Firm. *IEEE Internet Comput.* **2010**, *14*, 92–96. Available online: <http://www-cdr.stanford.edu/~petrie/online/peer2peer/vision2010.pdf/> (accessed on 13 January 2011).
4. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Available online: <http://www.w3.org/TR/wsdl20/> (accessed on 13 January 2011).
5. Dagstuhl Seminar on Service-Oriented Computing Session Summary Cross Cutting Concerns. Available online: <http://tinyurl.com/webservdef/> (accessed on 13 January 2011).
6. SA-REST: Semantic Annotation of Web Resources. Available online: <http://www.w3.org/Submission/SA-REST/> (accessed on 13 January 2011).
7. Petrie, C. Practical Web Services. *IEEE Internet Comput.* **2009**, *13*, 93–96. Available online: <http://www-cdr.stanford.edu/~petrie/online/peer2peer/practicalws.pdf/> (accessed on 13 January 2011).
8. Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. *Scient. Am.* **2001**, *184*, 34–43.
9. Petrie, C.; Hochstein, A.; Genesereth, M. Semantics for Smart Services. In *The Science of Service Systems*; Demirkan, H., Spohrer, J.C., Krishna, V., Eds.; Springer: New York, NY, USA, 2011, in press.

10. Petrie, C. The Semantics of Semantics. *IEEE Internet Comput.* 2009. Available online: <http://www-cdr.stanford.edu/~petrie/online/peer2peer/semantics.pdf/> (accessed on 13 January 2011).
11. Petrie, C. Pragmatic Semantics. *IEEE Internet Comput.* 2005, 9, 95–96. Available online: <http://www-cdr.stanford.edu/~petrie/online/peer2peer/w505.pdf/> (accessed on 13 January 2011).
12. How to Publish Linked Data on the Web. Available online: <http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/> (accessed on 13 January 2011).
13. This is a standard formula used in the formalization of planning of web services among, other actions.
14. Petrie, C.; Bussler, C. The Myth of Open Web Services—The Rise of the Service Parks. *IEEE Internet Comput.* 2008, 12, 94–96. Available online: <http://www-cdr.stanford.edu/~petrie/online/peer2peer/serviceparks.pdf/> (accessed on 13 January 2011).
15. UDDI/BSR Requirements Gap Analysis. Available online: <http://logic.stanford.edu/talks/gap/> (accessed on 13 January 2011).
16. RDF/XML Syntax Specification (Revised). Available online: <http://www.w3.org/TR/REC-rdf-syntax/> (accessed on 13 January 2011).
17. Introducing Rich Snippets. Available online: <http://googlewebmastercentral.blogspot.com/2009/05/introducing-rich-snippets.html/> (accessed on 17 February 2011).
18. Kassoff, M.; Petrie, C.; Genesereth, M. Semantic Email Addressing: The Killer App? *IEEE Internet Comput.* 2009, 13, 30–37. Available online: <http://logic.stanford.edu/sharing/Papers/sea-ic.pdf/> (accessed on 13 January 2011).
19. Petrie, C. Planning Process Instances with Web Services. Available online: <http://logic.stanford.edu/sharing/papers/IVIS09-serviceplanning.pdf/> (accessed on 13 January 2011).
20. Date, C.J. *What Not How: The Business Rules Approach to Spplication Development*; Addison-Wesley: Boston, MA, USA, 2000.
21. Genesereth, M. Stanford Logic Group, Stanford University, Stanford, CA, USA. Private Communications, 2010.
22. Malone, T.; Crowston, K. The Interdisciplinary Study of Coordination. *ACM Comput. Surv.* 1994, 26, 87–119.
23. Google Docs. Available online: <https://docs.google.com/> (accessed on 20 January 2011).
24. Kassoff, M.; Valente, A. *An Introduction to Logical Spreadsheets. Knowledge Engineering Review*; Cambridge University Press: Cambridge, UK, 2007; Volume 22, pp. 213–219.
25. Petrie, C. Collective Work. *IEEE Internet Comput.* 2008, 12, 80–82. Available online: <http://www-cdr.stanford.edu/~petrie/online/peer2peer/collectivework.pdf/> (accessed on 13 January 2011).
26. Kassoff, M.; Genesereth, M. PrediCalc: A Logical Spreadsheet Management System. In *Proceedings of the 31st VLDB Conference*, Trondheim, Norway, 2005; pp. 1247–1250.
27. Bertossi, L.; Hunter, A.; Schaub, T. Introduction to Inconsistency Tolerance. In *Inconsistency Tolerance*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–14.
28. Tolksdorf, R. A Dependency Markup Language for Web Services. In *Proceedings of Net.ObjectDays 2002 Workshop*, Erfurt, Germany, 7–10 October 2002; pp. 573–584.

29. Petrie, C.; Goldmann, S.; Raquet, A. Agent-Based Project Management. In *Lecture Notes in AI 1600*; Springer-Verlag: Berlin, Germany, 1999. Available online: <http://www-cdr.stanford.edu/ProcessLink/papers/DPM/dpm.html/> (accessed on 13 January 2011).
30. Petrie, C. The Redux' Server. In *Proceedings of International Conference on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, The Netherlands, 12–14 May 1993. Available online: <http://www-cdr.stanford.edu/ProcessLink/papers/redux-prime.pdf/> (accessed on 13 January 2011).
31. Petrie, C.; Webster, T.; Cutkosky, M. Using Pareto Optimality to Coordinate Distributed Agents. *AIEDAM* **1995**, *9*, 269–281. Available online: <http://www-cdr.stanford.edu/NextLink/papers/pareto/pareto.html/> (accessed on 13 January 2011).
32. Wielinga, B.; Schreiber, G. Configuration-Design Problem Solving. *IEEE Expert* **1997**, *12*, 49–56.
33. Haag, A.; Rieman, S. Product Configuration as Decision Support: The Declarative Paradigm in Practice. **2010**, in preparation.
34. ProcessLink. Available online: <http://www-cdr.stanford.edu/ProcessLink/> (accessed on 13 January 2011).
35. Fensel, D. Unifying Reasoning and Search to Web Scale. *IEEE Internet Comput.* **2007**, *11*, 94–96.
36. Petrie, C.; Jeon, H.; Cutkosky, M. Combining Constraint Propagation and Backtracking for Distributed Engineering. In *Proceedings of AAI'97 Workshop on Constraints and Agents*, Providence, RI, USA, 27–31 July 1997; Technical Report WS-97-05; AAI Press: Menlo Park, CA, USA, 1997.

© 2011 by the author; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>.)