Giuseppe Valetto

PhD Thesis Proposal, Columbia University

May 2000

Process-Orchestrated Software: Towards a Workflow Approach to the Coordination of Distributed Systems

1 Introduction

As the Internet turns into an essential business and production infrastructure, networked business and work in general become more and more common; for instance, virtual enterprises emerge as an innovative way to launch, organize and carry out initiatives, either within a large, global corporation, or as dynamic joint ventures between geographically dispersed organizations.

In such a context, the significance of distributed workflow (WF in the remainder) technology is steadily increasing. WF aims at the support of complex collaborative *processes*, composed of activities, in which the synchronization and coordination of the activities and the people having a part in them (the stakeholders of the process) is an essential characteristic. WF is based on the concept of a process model, describing the process to be followed, and on facilities (collectively termed the *process enactment engine*) for supporting and guiding the work of stakeholders according to that model. Distributed WF nowadays provides many of the necessary paradigms, techniques and tools that support the management of complex, dynamic and decentralized business practices, their stakeholders, and their processes.

There are a number of dimensions concurring to WF distribution, which interact - and sometimes conflict - with one other in various, complex ways, such as distribution of the WF information, the WF actors, tools employed to carry out the work, and the work itself. Distributed Workflow Management Systems (WFMSs in the remainder) deal with those dimensions in different ways, operating on and privileging some of the dimensions rather than others, depending on the paradigm and technology of choice, and complete decentralization of WF along all of its distribution dimensions is still a challenge.

On the other hand, approaches and techniques that have been established for and have become typical of distributed WF show great potential for more general applicability besides the classic application domain of WF, i.e. the support and guidance of decentralized business practices. There are a number of problems regarding the coordination of complex distributed systems and applications that require the execution of some kind of process, although the level of explicit process awareness held by the system components and/or stakeholders, and embedded in the overall system, may greatly vary. For these problems, concepts and techniques lifted from distributed WF and leveraging on its understanding and representation of coordination as a process can be fruitfully exploited.

Some of these problems are of great relevance for and recur often in state-of-the-art distributed systems. Two important, wide domains for coordination are the run-time monitoring and control of functional and non-functional properties of the components of a distributed system, and the dynamically determined cooperation within a group of software agents towards some common goal (for instance, the retrieval or production of some piece of information).

In fact, increasing convergence between the disciplines studying agent-based systems, distributed WF and coordination of component-based systems can be clearly detected, as discussed in more depth in Section 0, with ample overlapping of concerns and opportunity for synergy. The coordination of agent-based and component-based systems involves only marginally or not at all humans and collaboration between them, which is traditionally a major focus of WF. Therefore, those kinds of coordination can be considered as *reference problems* to assess the possibility to extend WF-like coordination techniques to distributed SW systems at large, and allow to clearly position the potential of distributed WF in that context.

The coordination of such systems provides a test bench for WF-based coordination techniques, which is at least as challenging and demanding as the domain of decentralized business processes, although with a very different focus. They present peculiarities that are not commonly specifically addressed by current WFMSs, e.g. completely automated process enactment. WFMSs, however, offer in principle adequate and important capabilities that are only partially present in other coordination techniques, such as high-level coordination models founded on the concept of an explicit process, and formalisms with both declarative and imperative connotations, which accommodate abstract analysis and reasoning as well as enforcement and guidance of the coordination model.

We intend to investigate the specific characteristics of the reference coordination problems mentioned above and how they can be tackled with and satisfied by some different models of distributed WF and the corresponding techniques, with any necessary adjustments and extensions. Given the nature and properties of the problems, the proposed distributed WF-based techniques for each of them may be very different, and also the definition of WF may vary, or be even somewhat stretched in some cases. A WF-based solution that aims to cover as much as possible all of the reference problems must necessarily be composite, and must operate over most (or even all) of the distribution dimensions of WF. Therefore, it can provide precious insights on those dimensions and their criticality and impact on the reference problems.

The final objective to evaluate the potential of distributed WF technology to handle the chosen reference coordination problems, and coordination of distributed SW in general: this includes not only identifying common traits and principles that can lead to a unified WF-based framework for the coordination of distributed systems, but also recognizing any outstanding idiosyncrasies that cannot be reconciled in such a framework, and either must be tackled with ad hoc solutions, or even stand out as open questions for research.

We intend to carry out our investigation in the light of a distributed systems case study, in which the reference coordination problems and business processes can coexist at different levels, i.e., coordination within a distributed collaborative application that is explicitly WF-aware, coordinated control of the functionality of that distributed application by a separate monitoring infrastructure, and coordinated interaction among the various components/agents that concur to the monitoring infrastructure itself. The case study is directed towards process-aware provision of multimedia services and information to groups of dispersed users carrying out teamwork, and includes the management of an advanced architecture for broadcasting and streaming video and audio, on the basis of available network and system resources. The case study can be also seen as a particular example within a more generic distributed system scenario, i.e. the *continual validation* of generic distributed applications.

The scale of distribution we intend to address is that of global dispersion of processes, SW components, stakeholders, and data. Thus, we are going to experiment with a set of techniques and mechanisms, which address those aspects at the global networking scale, and whose integration within a globally decentralized WFMS will help reconciling the various distributed dimensions of WF, as well as resolving the diverse distributed coordination reference problems.

In the remainder of this document, the many facets of this work are analyzed in further depth. In Section 2, we discuss WF distribution dimensions. In Section 3, we describe the set of distributed coordination reference problems and outline the WF-based techniques that can be exploited for their solution. In Section 4, we describe the continual validation scenario, underline the relevance of the various reference problems in that context, and propose our case study. In Section 5, we describe a set of mechanisms that we plan to employ to address the complexities inherent to the distribution coordination reference problems, via an enhanced distribution of WF along its various dimensions. In Section 0, we compare and contrast some distributed WFMS available as research prototypes and/or products, and we discuss how their characteristics could be exploited for our purposes. In Section 7, we discuss appropriate ways and principles for the evaluation of the potential of distributed WF paradigms and technologies with respect to the reference problems and coordination of SW distributed systems in general. In Section 8, we provide an outline of the various research issues that will be tackled in this work, with a tentative schedule.

2 Dimensions of distribution for decentralized WF

"Classic" *centralized* WFMSs, such as [56] [57] [58] [59] , adopt some variant of the client/server architecture, in which all WF data are persistently kept in a database of some sort, which usually sits with (physically or logically) and is administered by the server. This paradigm – which was in fact effective only

for WFMSs running on LANs and having a limited number of relatively clustered users and computer hosts, has been progressively challenged and is finally being broken due to the emergence of more decentralized computing frameworks, distributed information infrastructures and dispersed organizational and work structures, which took to utilizing the Internet as a commodity and the WWW as a paradigm.

Distributed WF services numerous scenarios and needs beyond the scope of centralized workflow –virtual enterprises, or (semi-)autonomous enactment of WF fragments at separate sites, or widely distributed large-scale projects, such as Open Source software development, to name a few – but is also way more challenging in terms of the organization and availability of dispersed data/resources, as well as in the coordination of activities taking place at multiple remote sites. In a widely distributed WFMS, the various WF agents in charge of tasks can be either human or computerized, and must cooperate and gather data and resources from potentially any point in the network. This can lead to various problems, especially on large-scale and heterogeneous networks, such as poor performance or even breakdown of the workflow or the workflow management system, depending on the availability of data and resources in given network conditions.

Those problems –among other things - break the assumption that all data and resources necessary to enact the WF are always readily and timely available to the enactment engine and/or its clients. That assumption has been inherited by centralized WFMSs and has not undergone yet any substantial revision. Due to the unpredictable QoS of the Internet and the load and complexity of the information exchanges involved, it hardly stands up already for most commercial state-of-the-art WFMSs, which provide to dispersed users Web-enabled clients connecting to a centralized server, and it is certainly even less valid for a multi-server or a truly decentralized WFMS that wants to operate at a truly global scale. In such systems, WF data and resources are potentially much more widely dispersed and the WFMS knows in the most favorable case only their location, via a pointer - e.g. a URL - but rarely owns them exclusively.

Some of the facets of WF distribution are: being able to fetch information as needed at the various stages of WF enactment, from locations far away or disconnected with respect to the location where the enactment takes place; publishing and making effectively available to the overall WFMS new data created by a WF task as its by-product; discovering, acquiring, or otherwise retrieving information that is available on the network at large and useful for the WF enactment but is not initially contemplated by the WF database (e.g. a set of documents resulting from a WF task performing an Internet search); managing in a coherent way all of this information, providing it with both a unified structure and different levels of presentation.

All in all, a number of interrelated distributions dimensions can be abstracted out from the list of issues above, and must be properly coherently and addressed. Two basic aspects are: how and where data is located, and how and where the architectural components of the WFMS (i.e. the WF agents) are deployed and activated. On top of these two major issues, another issue is how work is distributed (i.e. assigned) to the WF agents active on the network.

In the remainder of this Section we discuss in depth the issues regarding WF data distribution, WF architecture distribution and work distribution, and we will recognize the various separate dimensions of distribution within each of them. A visualization of the dimensions is provided in Table 1 in Appendix A. Before that, however, we take some time to define more precisely the *global networking* scenario we envision as our reference distribution framework.

2.1 Global networking distribution

We intend to address the problem space of WF distribution in the context of networks that are global and heterogeneous. As for globality, the model of reference is the Internet, which represents for today's technological endeavors both a challenge (first of all, in term of scalability of the architecture) and an opportunity (for example, providing commodity almost ubiquitous communication protocols and data exchange/storage facilities). However, it is necessary to underline that we do not intend to bind (and hence limit) ourselves to any Internet dominant protocols (e.g. HTTP) and paradigms (e.g. the WWW). In fact, in addition to the above distributed WFMSs must also operate upon and take in account a variety of protocols and facilities, such as various middleware platforms (see Section 6.3.3), which may be layered on top of the Internet, but represent in themselves high-level communication and distribution infrastructures. Therefore, it is not possible to assume and leverage homogeneous networking scenario for the distribution of WF data, agents and work items, which must me able to reside, relocate and communicate in a variety of network environments. For instance, a global network may incorporate some portions of the WWW, but also a

corporate Intranet, a distributed computing environment running some distributed application, a set of mobile terminals connected through different media. To ensure distribution of a WF across all of these and other network "fragments", interoperability mechanisms and "bridges" among them constitute a key issue.

The heterogeneity of a global networking environment is further emphasized these days by phenomena like spontaneous and active networking. Technologies like Bluetooth [76] will make portable networked computing and communication devices increasingly powerful and common, with the ability to host non-trivial application and to establish among those applications network connections on-the-fly. Active networks [104] will enable dynamic adaptation of the properties and protocols of nodes and connections on the network. Advancements of this kind are going to have a major influence upon any state-of-the-art distributed system, its architecture, and of course also its coordination. They also apply to distributed WFMSs and pervades all WF distribution dimensions, adding a whole level of complexity to them: not only the data, code, queries and computations of the WF may be distributed and move around a network infrastructure that is stable, but also the network itself may change, with nodes that appear, disappear and change their location, and connections that modify their characteristics in a dynamic way.

In this document, we consistently refer to the term *global network* to characterize a networking environment as the one sketched above, and to indicate the reference distribution framework for our work.

2.2 Distribution of WF information

A WFMS necessarily relies on a wealth of different kinds of information. The following loose categorization highlights two major kinds of WF data, i.e. data concerned with the specification of WF (design data) and data concerned with the enactment of the specified WF (runtime data):

- WF design data:
  - WF declaration: it represents the layout of the WF, in terms of plans, tasks, relationships among tasks, etc. For tasks, valid states and state transitions are specified, as well as the types of their inputs, outputs and resources; the relationships among tasks contribute to the control as well as the data flow. The WF declaration may be more or less explicit, depending on the formalism employed to express it; however, a minimal WF design unit must be explicitly present in any formalism, to allow for process construction and reasoning (e.g. task, action, speech act etc.).
  - WF definition: it is the behavior - i.e. the set of actions - associated with, each of the various WF tasks, or given task state transitions, or given transitions from task to task. Actions can be expressed in various ways, for instance with procedural code of some sort. Actions typically manipulate the task input (possibly via the invocation of some tool), produce the task output, change the state of the task, cause the enactment of other tasks, etc.

    Together, the WF declaration and the WF definition provide the *WF specification*.
  - artifact and resource types: they provide a model of the information domain on which the WF operates. The type system can be very sophisticated or very straightforward, depending on the WFMS and the application. Usually, artifact and resource types are associated with their own state information and with other metadata that is intended as a descriptor for the artifacts or resources of that type (i.e. ID, location, ownership, etc.). Types can also be sometimes created on the fly and added to the type system, or the type information may be embedded in the instances, achieving self-descriptive artifact/resources.
  - roles: categories of WF agents (either human or computerized). Tasks are assigned to agents on the basis of role; therefore roles can be seen as categories of resources in many ways.
- WF runtime data:
  - artifacts: documents or other information content manipulated by instantiated tasks. They comply with the artifact type system and follow the data flow of the WF. Each artifact is seen as an instance of an artifact type, therefore is associated by the WFMS to the appropriate metadata.
  - resources: in order to enact a task, the WFMS may have to secure a pool of resources. Resource availability is therefore a kind of task precondition. A resource can be any kind of entity that has a part in the work represented by the task and has some value or cost, normally because of its limited supply: typically, WF stakeholders, HW, SW tools, time, etc. Like for artifacts, resources are instances of resource types.

The problem of resource acquisition and availability acquires a particular relevance for a largely decentralized WF. In such cases, it is often impossible for the WFMS to have at start-up a complete mapping of its whole information and resource space. Some of the resources may not be under the direct ownership and control of the WFMS from the beginning, and must be looked after, discovered, retrieved and secured at various stages during the WF enactment, in order to carry out some tasks. Any WF-related element not known and owned by the WFMS at startup takes the role of a precious – although *impromptu* - resource, and its availability may heavily influence the data flow of the WF. Sometimes the WFMS may even have to produce new instances of impromptu resources – although they do not themselves take part in the WF objectives - in order to proceed with some task.

The acquisition of impromptu resources can be seen as a WF task itself, although perhaps as an ancillary, implicit, or "sideways" task with respect to the "main" WF.

This situation also contributes to somewhat blur the distinction between artifacts and resources. In fact, a prominent example of impromptu resources are *information impromptu resources.* They are data that are relevant to the WF at some point, but are not initially elements in the artifact repository, although they might be incorporated in it once acquired. Information impromptu resources are typically a significant issue for distributed WFMS of global scale, either because their goal is the production of information or knowledge out of and over the global network, or because they are superimposed on back-end dispersed computational environments that manage and publish over the network substantial amounts of information on their own, i.e., outside the control of the WF. Typical information impromptu resources are the result of an Internet search performed on the behalf of a WF agent (either human or computerized).

Among the categories of information impromptu resources there are:

- data whose location is not known, and must be looked after
- data that is not immediately publicly available for some reason, for which access must be granted.
- subsets/supersets of available WF data, because of their particular relevance to some of the workflow steps
- expensive/heavy remote data (in terms of bandwidth, access time, duration or other considerations)

The more the WF is decentralized, the more impromptu resources can be dispersed – even at the same scale as the global network- and the issue of securing them to enable enactment becomes particularly critical.

Notice that the impromptu resource concept can be generalized with respect to the task and process states, in case they are decentralized and a WF agent needs to retrieve an unknown fragment of the process state located somewhere remotely. That piece of information can be seen as a special kind of resource, which enables the control flow of the WF, rather than the data flow.

- task state: at any moment during the WF enactment, each instantiated task is in one of the possible valid states specified in its declaration. The task state evolves over time, on the basis of the work carried out in the task and other correlate tasks.
- process state: it can be either expressed as the combination of the states of all instantiated tasks in the WF at a given moment, or as a separate concept of state, representing in some way the "big picture" for the overall WF (e.g. level of completion, reach of intermediate or final goals, etc.)

All of the above information must be located, identified and organized appropriately; it also must be made timely available to the agents carrying out distributed WF, according to what specific WF data they need at any given enactment stage. One issue here is to circumscribe the amount of data that must be provided to each agent in the most effective way.

Distribution of WF information - as described above - defines at least three distribution dimensions:

- distribution of the WF specification – both declaration and definition (design data);
- distribution of WF state (enactment data);

- distribution of WF artifact and resources, including impromptu resources, according to their type (both design and enactment data).

When all of the three dimensions above are completely supported, in principle each type and each piece of data may be remotely distributed with respect to any other. For instance, the declaration of a WF fragment and the action definitions for the same fragment can be kept separated: the task declaration can for example be stored on a WWW server, while the corresponding action can reside somewhere else, perhaps even being a computation transported over the network by a mobile SW agent.

To handle this wealth of dispersed information an approach that consistently characterizes and organizes all the various kinds of information relevant to WF is needed. In Section 5.1, we will propose a solution that is based on the WWW as the common infrastructure for storing and making available the information.

2.3 Distribution of workflow enactment agents

A decentralized WFMS must support the enactment of WF fragments by a set of WF agents residing on different and potentially very dispersed nodes of the network. Besides the distribution of data, decentralized WFMSs must also address the deployment of WF agents.

We employ the term *WF agent* to indicate a computer application that enables interaction with other parts of the WFMS, with the purpose of enacting the WF. WF agents can act on behalf of *WF actors*, i.e., human stakeholders of the process; in the centralized WF model, WF agents represent clients to the WF enactment engine, and human stakeholders are their end users. For a decentralized WFMS, a rigid distinction between a server accommodating the WF enactment engine and the WF agents as its clients hardly exists anymore. Depending on the degree of decentralization, some or even all of the WF agents may be provided with WF enactment capabilities such as lightweight enactment engines, enhancing their autonomy. Thus, agents may be able to execute WF fragments on their own and exploit whatever communication infrastructure is available for the necessary notifications and coordination with other agents: for instance, an agent may fetch a task declaration from the location where it is stored, initiate its enactment, fetch and execute the corresponding action(s), in the meanwhile acquiring and employing any necessary artifacts and resources. In a decentralized WFMS scenario, a WF agent could be at the same time a server and a client for other agents. This also allows to accommodate particularly well the concept of purely *computerized WF agents* that enact WF fragments without human supervision. Notice that computerized agents can also be present in the centralized case, although this implies a master/slave relationship between the enactment engine and those agents; in a decentralized WFMS, they can be instead largely autonomous.

The architectural distribution of WF agents has in the first place a static dimension, reflecting the deployment of the WFMS at the time of WF enactment; that dimension is supplemented by a dynamic dimension reflecting modifications of the initial architecture, due to the instantiation of new agents, or the migration of existing ones to new hosts in the network, etc.

As far as the static architecture is concerned, a typical paramount factor is the allocation of WF agents to any existing human actors, which effectively enables remote cooperation among (groups of) human users. The role of these components, even in a decentralized WFMS, is still mainly to provide users with an interface to the WFMS functionality, similar to clients in traditional client/server centralized WFMS, but without that kind of architectural constraints. Some computerized WF agents can be opportunistically deployed at enactment time, for instance to enable the enactment of WF steps at relevant network locations that are remote with respect to all the existing stakeholders; they thus take part in the static distribution dimension, in the same way of other components of the distributed WFMS designed to provide some services or utilities to agents and the overall system.

Computerized WF agents take also a major part in the dynamic dimension of the architectural distribution of a WFMS, since they can be deployed on the fly as needed. Even more so in case computerized agents are also mobile software [75], which enables them to physically follow the control and/or data flow through multiple WF steps that are enacted on different hosts, and execute them. Mobility is also of interest for WF agents allocated to human users, in case disconnected operation is supported by the WFMS.

The presence of a dynamic architectural dimension of WF demands for means to express the directives that modify the layout of the WF agents either by deployment of new agents or mobilization of existing ones. These directives are necessarily correlated with the state of the WF being enacted and implement an internal coordination mechanism for the decentralized WFMS.

2.4 Distribution of WF enactment

The distribution of WF enactment (i.e. task instances) to WF agents is an issue that must be addressed by any WFMS, either centralized or decentralized; the main difference in decentralized case is once again due to the fact that there is no clear-cut distinction between those components of the WFMS in charge of distributing the work and those that receive work.

WF enactment is distributed to WF agents according to two basic modalities: tasks are either pushed to or pulled by any of the various distributed agents. Hybrid modalities, such as a state server that works as a shared blackboard – like in ProcessWall [60] - work essentially as intermediaries, decoupling the components of the WFMS that push tasks and those that pull them.

The pull modality is the one typically employed by WF agents allocated to human users, since it implies the voluntary selection and uptake of one of the various possible tasks that can be legally enacted at any given moment. Task pulling may also occur on the part of computerized WF agents, for instance when an agent is in charge of a WF fragment that includes a number of tasks, and tries to execute all of them in sequence. With the pull modality, the WFMS assumes a *reactive* behavior.

Pushing of WF tasks occurs due to either *automation* or *delegation*. Delegation occurs on an agent-to-agent basis, either because of the decision of a knowledgeable human user, or because of some suitable automatic delegation policy. Automation takes place on the basis of changes in the state of the WF and can be directed to any agent (human or computerized) that can be requested to enact a given task. The push modality – and automation especially - provides the WFMS with *proactive* behavior, in addition to reactive [112].

The balance between the reactive and proactive dimensions of WF enactment distribution depends on the WF paradigm, the features of the WFMS and the characteristics of the process to be enacted.

2.4.1 Task pushing

Task pushing implies that the work is assigned to the WF agent by some other external component of the WFMS, which – by delegation or automation - indicates the receiving agent and binds data and resources to the task on its behalf. In a centralized WFMS, tasks are simply pushed to suitable clients by the server – which runs the WF enactment engine and therefore holds all the necessary knowledge. In a decentralized WFMS, instead, task pushing schemes are necessarily more complex, since some or even all WF agents can act both as an enactment engine and a client. In the case of maximum decentralization, which implies non-hierarchical and peer-to-peer relationships between the WF agents, any of the dispersed WF agents can in principle push a task to any other. Various coordination schemes can be borrowed from models for organizing communities of generic SW agents - such as organizational structuring, contracting, multi-agent planning, or dynamic negotiation [81] - and used for task pushing in a decentralized WFMS.

The proactive dimension of work distribution is the most dependent upon considerations about data and architecture distribution. In the ideal conditions, a task is always pushed to an agent that is available and capable to carry it out, and can easily fetch all the information and resources necessary to lead to the completion of the task. In practice, it can happen that the access to the needed information and resources is extremely costly or difficult, or even impossible depending on the configuration of the architecture and the dispersion of the data and resources. In this case, the proactive capabilities of a WFMS could become a source of inefficiency for the WF. To overcome this problem it may be necessary to modify either the distribution of the WF agents (by deploying/instantiating/migrating an agent to a host more convenient for the purpose of carrying out the task), or the distribution of data, thus making available (i.e. transferring/copying) to some existing agent the necessary data and resources.

Since both solutions can be themselves costly and complex, it would be extremely important to be able to predict when adjustments to the data and/or architecture distribution are likely to be needed, in order to preempt those situations by "working behind the scenes" in preparation for them. To do this, a *task forecasting* mechanism must be associated with the proactive facilities of the WFMS. The importance of task forecasting coupled with proactivity is that it enables "smart" policies that are data/resource-aware for agent instantiation or migration, as well as for enhanced availability of data and resources to agents. This can lead to improved efficiency in the distribution of work and be a big help in resolving the complexities caused by wide dispersion of the WF along the data and architecture dimensions.

2.4.2 Task pulling

As for the reactive dimension of work distribution, when an agent chooses to pull a task, the WFMS reacts by providing it with all the necessary specification information about that task. Again, while in centralized WF reactivity is achieved simply by having the server responding to client requests, in decentralized WF reactivity is subtler since any WF agent may take up the double role of ”server” and “client”: for example, a WF agent can even react (implicitly) to some WF enactment request raised by itself. Also, task pulling among a group of computerized WF agents typically involves an appropriate model of ineter-agent coordination that must be adequately embodied by the WFMS.

The binding of input artifacts and resources of the right types to a pulled task is part of the reactive behavior and can happen in various ways: for example, it may be explicitly specified by the agent when pulling the task, or may be implicitly resolved depending on the state of the task, the overall WF and the artifact/resource pool, etc.

Bound artifacts and resources are either local or remote with respect to the WF agent in charge of the task. Remote fetching can be carried out in various ways, such as obtaining a remote lock (as in WebDAV [29]) or by making local work copies that must be reconciled with the remote master copy at the end of the task, according to some consistency model embraced by the WFMS (such as extended transaction models [108], divergence control [114], etc.). Once the binding process is complete, any action corresponding to the task must be also fetched and executed.

The reactive dimension of work distribution does not particularly interfere with the data and architecture distribution dimensions, unless the request of pulling a task by a WF agent is hindered by the relative location and availability of the agent and the WF data: the only possible workaround is to redistribute the WF data in a more efficient way. Again, task forecasting capabilities would allow to prepare for the occurrence of these situations and enhance the availability of WF data to agents that may need it.

2.5 A synopsis of WF distribution characteristics

We propose hereby a way to schematically identify the various characteristics of a decentralized WF, in particular with respect to distribution. While the categorization of the distribution dimensions - as it has been discussed earlier in this Section -is sufficiently systematic and clearly disjoint to allow for this kind of schematics, it is certainly more difficult to come up with scales for those dimensions. In fact, we maintain that an attempt to elaborate anything like an orderly sequence of distinct “values” within some spectrum, to indicate degrees of distribution along one or more distribution dimensions, would be quite artificial and thus hold little value. For this reason, the dimensions are going to be described by a scattered set of concise attributes/phrases referring to the characteristics recognizable in the coordination problems under our scrutiny and their corresponding WF. The attribute sets are therefore not decided a priori at this stage, but will emerge as a result of the discussion about the various coordination problems of interest. The attributes will help positioning each WF along each dimension, but first of all with respect to each other.

In Table 1 in Appendix A, we show the schematics of the synopsis, leaving a certain number of empty spaces that will be filled by attributes of the various kinds of WF addressing the coordination problems. Notice that, for the sake of completeness, we also intend to include in the synopsis some other prominent properties of each WF, which are not directly related to distribution, but are helpful to characterize it (e.g. the explicitness of the process, its repeatability, etc.).

The synoptic tables and the complete prospect that follows the analysis of the coordination problem we mean to address, are found in Appendix A.

3 WF-based techniques for some complex distributed coordination problems

In this Section we analyze separately the peculiarities of the distributed coordination problems of ,, i.e. how to control at run-time a generic external *target system*, and how to orchestrate the cooperation of a community of SW agents in an agent-based system. For each of them, we then discuss the characteristics of their inherent coordination processes, and how they fit with respect to the various distribution dimensions of WF. While doing this, we identify what kind of WF techniques can be suitable to provide a WF-based solution to the coordination problem.

3.1 Coordination problem 1: run-time control of a distributed system

In run-time control of distributed systems, the coordination problem consists in carrying out the adaptation and re-configuration of the structure and behavior of the system to be controlled, in response to conditions that occur within the target system, or over the network, or whatever other communication and computational infrastructure the system relies upon.

In this coordination problem, humans may be completely absent from the controlled system, or – if present - may simply cover simple and punctual (although potentially critical) decisional or authorization roles. Therefore, the coordination model does not need to account for support and guidance of the intensive, creative, open-ended and long-lived activities typically associated to human work. It rather enables the execution of automated imperative directives, with the purpose to modify the functioning parameters of the target system.

The elements participating in the coordination are primarily or exclusively software components, such as:

- the components of the system that must be controlled;
- *probes*, which check for some conditions to occur in the target system and report those occurrences
- events/notifications that represent the conditions reported by the probes; single events in themselves may or may not carry enough relevant information to enable the exertion of control directives; meaningful notifications might emerge only from the occurrence of certain partially ordered sets - *posets* - of events, composed by means of appropriate posets recognition mechanism, such as those in [54] [55]. For the sake of brevity, however, when in the remainder we use the generic term "event", we indicate some notification (possibly composed out of a poset) that already holds meaning for the control of the target system.
- active components that actually provide some form of control onto the original system, directly or via external programs, tools, or components.

An informal view of how the elements above interrelate is provided in Figure 1 (see Appendix A).

Probes and active components must be deployed in such a way to conveniently superimpose a complete control infrastructure over the architecture of the target system. This implies that the architectural distribution of the WFMS carrying out the control WF is influenced by the architecture of the target system, and in general the architectural styles of the two must somehow match, or even strictly correspond.

Purely reactive WF techniques can be effectively used in the control WF context to integrate the components of the original system that must be controlled, the probes and the active components in a relatively tightly coupled fashion. Both the data and control flows originate in the form of events from the probing components and are processed by the active components, which constitute the actual enactment engines for the control WF: whenever some event recognizable by any of the active components is reported, a reaction is fired, which produces some consequences on the target system, such as modifications of its structure or behavior. Such a reaction can have effects that are distributed over a multiplicity of components of the target system. The WF specification in such a case is very fragmented, and consists simply of the description of the set of monitored conditions and the definition of corresponding reactions.

From the conceptual point of view, the specification of a purely reactive WF of this kind - independently from the actual WF description formalism supported by the WFMS enacting it- can be seen as a set of Event-Action rules, or some variation thereof. In fact, the Event-Action paradigm is based on rules whose left-hand side (the Event) is a declarative description of a pattern that defines a situation of interest, while the right-hand side (the Action) is an imperative program to be performed when that situation occurs. Actions can have side effects with respect to the WF, that is, cause the emission of other events. This matches perfectly well the purely reactive behavior described above.

The Event-Action paradigm can be augmented in various ways: for instance, the basic paradigm is practically stateless, but it is possible to introduce some notion of state in the reactive system, and move towards an Event-Condition-Action (ECA) [77] [78] rules paradigm. Conditions are predicates over the state of the system – as well as the content of the received event - which must be somehow available and known to components that receive events and execute actions: only if the condition attached to a matching rule is verified, the corresponding action gets fired. Another enhancement is to add *Alternative Actions*,

moving from the ECA to the ECAA rules paradigm, which allows defining actions that are fired in case the condition of a matching rule is NOT satisfied.

For a purely reactive WF that embraces the Event-Action paradigm, it may be difficult to lay out a priori a process model. However, the kind of coordination required for controlling a complex distributed system can in general be achieved only through orderly sequences of concatenated reactive steps. Therefore, the WF specification may be constructed in a bottom-up fashion, as reaction patterns and chains that make up WF fragments, possibly carried out by multiple dispersed active components. Or the overall process is even recognized and derived only a posteriori, by analyzing activity logs of the control agents (like in the approach of Balboa [110]). In fact, since the set of conditions that enable the firing of reactions is limited, also the kinds of control loops provided by those reactions into the target system are bound to be similar over time, resulting therefore in a highly repeatable - although rather implicit – process, which can be analyzed and extracted from practice, perhaps to achieve incremental process improvement, via tuning of the various rules.

The specifications of the control WF, be it in the form of Event-Action rules or in some appropriate other formalism that captures its essentially reactive nature, may be distributed in any way among the active control components, which are sensitive to notifications by the probes and are the computerized WF agents. Each WF agent may own any subset of the overall set, i.e. any portion of the complete WF specification.

In case the control system needs to keep track of some form of state in order to exert control over the target system (i.e. the chosen formalism is equivalent to ECA or even ECAA rules), such WF state must be made available to all the WF agents, via either full duplication at all the agent locations, or full distribution of the state information over the network, in a way that guarantees its complete accessibility as needs be.

Notice that this distributed coordination problem is not in general particularly involved with the manipulation of artifacts. Artifacts are not a major concern of the WF and the extent to which artifacts take part in the control process varies substantially, depending on the nature of the target system, and the kind and means of control employed by the controlling system. Resources are much more crucial than artifacts in this kind of WF: in fact, the resources for the WF are in the first place the components of the target system, plus any kind of external tool or SW component that is used by the WF to implement any necessary control procedure onto the target system. Normally, impromptu resources are not crucial, since the process is not particularly open-ended, nor varied in its activities, and (as for information impromptu resources) the production or manipulation of large amounts of information is not the primary goal of the WF, although it might well be that of the controlled system. In a way, however, the complex notifications composed from raw probe events may be seen as information impromptu resources.

3.1.1 Relevant distribution dimensions

The relevant entities that are to be distributed in the reactive control WF are the following:

- WF data
  - Design data
    - WF specification
      - Declarations of events of interest and associated reactions can be unified and colocated: a number of rule[1] repositories (rule hosts) can be dispersed over the network and pointers (e.g. URLs) to the various rules are provided to WF agents in charge of them.
      - Definitions of the imperative programs implementing the WF-guided reactions can be placed remotely with respect to the rules. They can be hosted on hosts different from rule hosts, as mobile code that can be downloaded and plugged into the control agents whenever a rule is enacted. Pointers from the rule declaration on a rule host to the corresponding coding of the action must be provided.

[1] For lack of a better concise term, we call them *rules* in the remainder of this Section; the reader should however be warned that the term rule does not imply here any exclusive endorsement of the Event-Action rules paradigm or some of its variations as the way to express the reactive distributed control WF.

        - Artifact, resource and role types: the type system can be stored in one or more type repositories which can be accessed by the WF agents whenever feasible.
    - Run-time data
        - WF state
            - task state: an enacted task corresponds to a reaction whose imperative program is being executed; therefore, the state of a task is expressed as the information about the corresponding rule and is owned and maintained by the agent in charge of that rule.
            - process state: the state of the overall WF is in this case more than simply the composition of the states of all the tasks in execution. It also includes the set of events circulating at any given moment. It is possible to maintain the events persistent via an *event repository*, which intercept and log all events for the sake of state inspection, history and recovery. Also event repositories can be distributed. Moreover, – in case a global state for the system is maintained and can be predicated upon by rules (as in the ECA paradigm)-, a suitable representation of state variables must be stored in one or more *state servers* and queries by agents evaluating rules' conditions must be directed to them in the most convenient way, with mechanisms that maximize availability and response performance.
        - Artifacts: artifacts of interest to the control WF are likely to be a by-product of the data (or control) flow of the target system, which need to be consulted by the WF for some reason. Therefore, they tend to be as distributed at least as much the target system.
        - Resources: they also tend to be as distributed at least as much the target system. Its components can themselves be seen as resources for the WF, as are configuration programs, SW utilities and other facilities whose main purpose is operating on the target system in support to the WF agents exerting control upon it. In some cases, resources may also be downloaded or operated from other hosts, which are remote to the WF architecture as well as to the target system architecture.
- WF architecture
    - Static architecture: the decentralized WFMS must provide a control infrastructure (i.e. probes plus active controllers) overlaying the architecture of the target system. Hence, the distribution WFMS architecture is strongly correlated to that of the controlled components of the target system.
    - Dynamic architecture: in general, the dynamic modification of the WF architecture is required only as a consequence of a variation in the architecture of the target system. For example, in case target system's components are instantiated or shut down on the fly, or due to component mobility, or in case the network hosting the target components changes its own topology, as in the case of spontaneous networking.
- WF enactment
    - Reactive enactment: as remarked earlier, the control WF is purely reactive, on the basis of event recognition. One issue here is how to assign responsibility for the various WF fragments to the control WF agents, i.e., which agents must be sensitive and respond to what events. A spectrum of options is available, from strict separation of concerns among the various agents (which implies specialization of each agent towards certain tasks) to complete replication of the set of interesting events among all the agents (which implies complete non-determinism in the matching and "consuming" of events and in the consequential execution of rules). Whatever is the choice, notice that, in order to respond to the stimuli provided by the target system via the probes, agents must be aware of assigned rules and in particular of their event profiles in advance with respect to their enactment. The only part of the WF specification that can be actually pulled on the fly are the reaction pograms, e.g. in the form of plug-in code to be executed by the WF agents.
    - Proactive enactment: it is mostly irrelevant, given the pure reactive nature of the control WF

Figure 2 shows in a rather informal fashion a hypothetical scenario for the distribution of the control WF and its corresponding WFMS, according to the various distribution issues discussed above. The approach taken in drawing the scenario is that of maximum distribution along all of the relevant dimensions. Such an approach is intended purely as a device for theoretical discussion about the decentralization of this kind of coordination process, and does not hint to or endorse any particular architectural decision. In fact, less

decentralized approaches – which are subsumed by the presented scenario - are also possible and may be even more feasible and efficient, depending on many factors, such as application-dependent considerations.

A synopsis related to the system control WF is provided in Table 2 (see Appendix A).

3.2 Coordination Problem 2: dynamic coordination of distributed cooperative computations

Many state-of-the-art distributed systems are organized and operate as a group (or community) of semi-autonomous distributed objects or components, generically referred to as *SW agents*. Each agent has its own properties and makes available to the group a set of computational capabilities, which may be very different from those of other agents. Agent communities carry out distributed computations by establishing at run time cooperation between the various agents in order to achieve some overall result or goal. Among the typical goals of agent-based systems, there are the retrieval or the production of some complex piece of information, decision support, etc. Goals of the agent community and ways to reach them can be expressed in many ways. Generally speaking, the cooperation among the agents towards their goal is carried out via a series of agent-to-agent interactions, with agents requiring services to each other on the basis of their current knowledge about the other agents' capabilities, their state and the state of the distributed computation that must be carried out.

Strategies (or *plans*) to reach the final goal are – depending on the underlying coordination model - either superimposed a priori over the agent community by some external "master coordinator", or decided among the agents, according to some self-organizing scheme of the community [81]. Inter-agent cooperation schemes are often described declaratively, or through some form of script. WF is another possibility, since it provides several means to organize and control the cooperative work of a group of largely independent entities: it expresses plans for reaching the goal as an explicit multi-participant process, indicates in a proactive way what work stages must be executed at a given moment, and handles in a reactive way events and situations (including unexpected ones) that occur in the course of the cooperative work. The specification of an agent coordination WF often takes the form of a plan towards a goal, which can be decomposed into sub-plans, i.e. WF fragments, implementing sub-goals (such as in JIL [18]).

A WF for the coordination of an agent-based distributed system has some peculiarities. It is completely automated– humans have usually no role at all in it, except possibly as the owners and initiators of the process. The SW agents participating in the community can be seen both as WF agents and as pools of resources (i.e. the capabilities and services they offer, as well as the information describing them). Moreover, the overall scope of the goals and the distributed computations implemented by agent communities are normally clearly defined from the start and limited: for instance the production of some information upon request. Such a WF does not need to be very open-ended, but has typically a limited (although not necessarily short) duration while it converges toward its final goal. On the other hand, it may be enacted repeatedly a large number of times, since the same computation can be requested over and over again to the distributed system made up by the agent community (e.g. a collaborative information search).

The richness of the WF specification essentially depends on the degree of expressiveness that the agent components can handle as their coordination language. Substantial amounts of complexity for the WF can be deferred from the design time to the enactment time, by relying upon autonomy and discretionality of the agents for the implementation of the coordination process. In that case, the WF specification may be simple and structurally straightforward, and it may describe the coordination process only at a high level of abstraction. Consequentially, the dynamic aspect of the WF (i.e. the way it's enacted from one process instance to another) may be extremely variable, in accordance with the high degree of discretionality of the agent community and since there may be a large number of cooperation patterns leading to the same high-level goals, i.e. instantiating the corresponding WF.

Noticeably, there is a tension between the level of coordination guidance provided via WF to an agent community and the level of autonomy of the agents. A WFMS orchestrating a community of agents would lean towards some form of externally imposed coordination, but must reconciliate this tendency with the self-organizing capabilities of the community. One extreme is *absolute guidance* by the WF, with proactivity on the part of the WFMS taking over and leaving very little or no autonomy at all to the agents. This is particularly evident in the case of a strongly centralized WFMS, in which the WF engine (the srever) takes the roles of the external master coordinator, and sees the various SW agents merely as

executors of its directives. Notice that, as the dispersion of the enactment responsibility among multiple WF agents increases, so does the blurring of distinctions between clients and servers, making absolute guidance difficult to achieve and possibly also inconvenient.

The opposite extreme is *full autonomy*: the WF only describes the overall process – possibly in very general terms - and the WFMS limits itself to serve as the process specification repository and as reactive run time support to the agents, which enact the WF relying uniquely on their self-organizing policies.

As an example of the many possible trade-offs between those extremes, one could imagine to organize and express the overall WF as a hierarchy of sub-processes: each of the agents may be proactively put in charge of some sub-process by some WFMS-run coordinator, but autonomously pursues any task taking part in that sub-process, and therefore the corresponding WF sub-goal. Interrelations (such as dependencies or precedence) between sub-processes must be resolved by the WFMS coordinator.

However, the one above is merely an example, and how to reach the most effective trade-off between autonomy and guidance (that is, in terms of an agent coordination WF, between reactivity and proactivity) depends on the characteristics of the cooperation policies built in the agent framework, as well as on the WF paradigm and the WFMS of choice, and it remains one of the major questions to be solved in order to adequately exploit WF techniques to coordinate a group of distributed agents.

The notion of an accessible distributed state is paramount in any agent-based system. Such a distributed state is founded on the composition of the states of all the SW agents in the community, but must also include information about the advancement of the various parts of the distributed computation carried out by the agents. As for an agent coordination WF, both of those aspects take part in its own concept of state and both must be globally accessible to the WF components. Therefore, they must be published and distributed by agents that own that kind of information.

All data used and produced by an agent-based system constitutes the artifacts for an agent coordination WF. Notice that such artifacts may be not only documents in the traditional sense, but also streams of data exchanged and processed on the fly by the cooperating agents. Such streams are in fact very volatile artifacts and can also be considered as resources (precisely, impromptu information resources) for the WF. Other kinds of impromptu resources for the WF may be identified and fetched by some specific SW agents, which have a data retrieval role in the agent-based system, rather than a data processing role, and might not be considered in the WF as full WF agents.

3.2.1 Relevant distribution dimensions

The relevant entities that are to be distributed in the agent coordination WF are the following:

- WF data
  - Design data
    - WF specification: declarations of WF tasks and the corresponding actions are most likely co-located, since they express goals and spell out plans to reach those goals. The way the information about the WF specification is distributed greatly varies depending on the coordination model embraced by the agent community and implemented by the WFMS. According to some models, (sub-)plans are exclusively owned by and known to appointed coordinating agents, while according to others the information is shared by all agents via either access to a plan repository or duplication of the information at the agent sites. However, the most flexible and dynamic agent coordination schemes account for the composition of the global plan in a bottom-up fashion, and require that different agents own and know certain sub-plans and exchange that information with the others as needed. A way to make this possible is to distribute the corresponding task decomposition hierarchy of the agent coordination WF on a number of hosts, keeping track of the composition relationships via pointers, and to provide those pointers to the various agents to enable them to acquire knowledge about the specification fragments. Notice that there is an issue here, related to the assignment of certain sub-plans and the corresponding sub-goals to specific agents, which actually creates a dependency with respect to the proactive dimension of WF enactment distribution, since the agent owning the specification information for a WF fragment is also those who are put in charge of its enactment.

        - Artifact, resource and role types: the type system can be stored in one or more type repositories which can be accessed by the WF agents whenever feasible (e.g. at initialization time).
    - Run-time data
        - WF state
            - Task state: a task is a sub-plan in execution by some agent and its state is owned and maintained by that agent.
            - Process state: the state of the overall process is represented by the composition of the execution states of all the various sub-plans, including the information on the achievement of the various sub-goals contributing to the final goal of the process. It is therefore owned collectively by and distributed together with the agents in charge of the various WF fragments. In case some global image of the process state is needed, the agents must be able to publish/communicate the state information they own to that global state server. One way to organize such a state server in a decentralized way can be per example with pointers from and to the state server to the fragments of state maintained by each single agent.
        - Artifacts: those that are taken as input of some WF task can be in principle distributed anywhere, but tend to be distributed as much as the agent (i.e. the WFMS) architecture: if the agent system supports mobility it is possible that agents are moved where those artifacts are located, otherwise the artifacts might be re-located onto the hosts where agents run. Also those artifacts that are produced as by-products of the execution of some WF task by an agent tend to be as distributed in the same way as the WFMS architecture.
        - Resources: as for artifacts, although they can be in principle distributed anywhere, they tend to be co-located with and distributed in the same way as the WFMS architecture. Notice that impromptu resource may have a significant role in various ways in this WF. For example, an agent may need to identify on the fly some other agent whose capabilities are needed to carry out some task; this is an example of the acquisition of an impromptu resource. Also, the dynamic data flow streams of information exchanged, manipulated and transformed by the agent community qualify as information impromptu resources.
- WF architecture
    - Static architecture: The architecture of the WFMS coincides with that of the agent community, since the SW agents are at the same time WF enactment engines. In case some components of the WF architecture cannot be reified as a participant in the SW agent community (e.g. a state server, see above), those components can be in principle deployed everywhere on the network, although the topology of agent community is likely to somewhat suggest their optimal locations., for instance because of proximity to WF agents that need them.
    - Dynamic architecture: The WFMS architecture is as dynamic as the SW agents are, again because the identification of SW agents as WF agents of the distributed WFMS.
- WF enactment:
    - Reactive enactment: task pulling can happen as the result of negotiating cooperation among some SW agents, e.g. a WF agent accepts to carry out a task as a response by a request by another agent.
    - Proactive enactment: proactivity by delegation and automation can be both supported. Delegation again happens as the result of some negotiation. In fact, since in any context - apart from the absolute guidance scenario outlined above - the SW agents that must be coordinated are also the WF agents enacting (fragments of) the coordination process, what looks like proactive behavior from the point of view of the delegating WF agent, may appear reactive on the part of the other agent. Automation can occur either internally to a WF agent pursuing a sub-plan, or across agents, depending on their task dependencies and the coordination model adopted by the community.

Figure 3 shows a hypothetical scenario for the distribution of the agent coordination WF and its corresponding WFMS, according to the various distribution issues discussed above. The approach taken in drawing the scenario is once again that of maximum distribution along all of the relevant dimensions, for the sake of theoretical discussion about all the distribution characteristics of this WF.

A synopsis related to the agent coordination WF is provided in Table 3 (see Appendix A).

4 The continual validation scenario

The continual validation scenario provides us with a unified context for experimenting with different distributed WF conceptual models and techniques that address the types of distributed coordination we are interested to investigate.

Continual validation is about run time monitoring and controlling of heterogeneous components taking part in some complex distributed system, in order to ensure that the system keeps operating within its declared valid functional as well as extra-functional parameters, via some appropriate distributed infrastructure (also called *meta-architecture*). The meta-architecture is in charge to detect the occurrence of certain conditions within the target system and to respond by modifying the run-time configuration and behavior of the target system's components, effectively creating either a feedback loop (to recover the system after it has reached a malfunctioning or otherwise undesired state), or a feed forward loop (to take preventive measures that preserve the system from degenerating into a malfunctioning or undesired state) into the target system.

Both the monitoring activities carried out within the meta-architecture and the controlling (feedback and feed forward) activities carried out by the meta-architecture upon the target system have – for any non trivial case – a considerable level of complexity, and can be seen as distinct but interrelated processes.

More specifically, the coordination of the monitoring entities that are part of the meta-architecture takes the form of a process focused on the production of notifications and reports upon the state of the target system. Such production of information involves multiple monitoring entities, each employing and handling a peculiar set of capabilities and data. Moreover, while the overall scheme for the production of the monitoring information may be fairly well understood and straightforward, the run time interactions between the monitoring entities must largely be decided on the fly. All of these considerations lead to consider the monitoring functionality of the meta-architecture as a case of orchestration of a community of largely autonomous but cooperating SW agents that produce streams of information, and opens the way to employ a WF with the corresponding characteristics - as discussed in Section 3.2.

On the other hand the feedback and feed forward control loops that must be enforced by the meta-architecture over the target system clearly provide an example of the coordination reference problem dealing with the external run time control of a distributed system, and can be addressed with a WF approach that has the corresponding characteristics, as discussed in Section 3.1.

The interplay between those WFs defines a more complex, federated WF, which guides the overall behavior of the meta-architecture as a whole, which therefore can itself be seen in many senses as a WF-coordinated distributed system.

In principle, any kind of distributed system can be the target of continual validation by an infrastructure that is able to enact the monitoring and control processes. However, a special and particularly intriguing case of continual validation is that in which the target distributed system itself enacts yet another independent process, i.e. is or incorporates a distributed WFMS running some decentralized WF. While the value of continual validation is not limited to this kind of cases and is in fact largely independent from the nature of the application being controlled, this special case provides especially interesting insights, since it supplies a unifying context for a very wide range of distributed WFs, in which a traditional usage of WF at the *application level* (that is, in the target system), which may contemplate the presence of human WF actors, coexists with the WF-based coordination of computerized distributed systems needed at the continual validation *infrastructure level*.

Notice that the choice of a WF-aware system as the subject of continual validation may also shed some light on considerations about the *reflectivity* of the usage of WF to coordinate distributed systems. Since a distributed WFMS with its components is in principle a valid subject for distributed systems coordination, it is arguable that the effective coordination of a generic distributed SW system via WF and along the various WF distribution dimensions is analogous to the coordination of the WF enactment architecture itself, and that must be possible to address the former and the latter in a unified way.

4.1 Case Study: AI2TV

We propose hereby a potential case study that enables the investigation of the continual validation scenario and all the WF facets involved in it, since WF is in fact present both at the application and at the infrastructure level. Furthermore, the case study represents a project that has relevant research connotations *per se* in a number of areas, among which there are Internet-scale distributed systems and WF.

AI2TV stands for "Adaptive Internet Interactive Team Video". The project aims at creating a collaborative virtual environment for group work, which includes an infrastructure for the provision of multimedia content relevant to the work carried out by the group, such as audio/video recordings of group discussions and decisions, informational and educational events, etc. Multimedia provision is mediated in AI2TV by the knowledge about the work context of the team and the individual members, which is derived in turn by the planned and enacted WF they follow. Multimedia is hence treated as both a new type of artifact and an additional resource (albeit an "expensive" one that must be carefully organized and managed, especially in an environment that demands for rapid and random access to it) for the WF of the distributed team.

The group members can be widely dispersed over the Internet, and may enjoy any combination of connectivity, ranging from 28.8k modem, to DSL, to cable to T1 or T3 lines. The multimedia content must hence be delivered over heterogeneous Internet links to heterogeneous platforms, taking in account widely varying bandwidths and QoS, in an *efficient* and *adaptive* manner. For instance, one of the requirements for the provision system is the support for synchronous watching of a video clip by all members of a team across their different equipment and networking capabilities, including support for video operations like fast forward, rewind, seek, etc.

To achieve its goals, the AI2TV provision infrastructure must:

1. First of all, derive a semantic structure from the multimedia information, which can be associated to the structure and semantics of (fragments) of the team's workflow. This way, multimedia provision may be adapted to the planned as well as the enacted work context of the team, by prioritizing the transfer of those segments that are deemed relevant.
2. Exploit any available heuristic perception or knowledge of present and anticipated user activity, in order to resolve connectivity and capability idiosyncrasies with a combination of content prefetching and caching, thus responding efficiently - even preventively - to users' requests.
3. Integrate a number of particular server cluster configurations, protocols, proxies, local client caches, and video management schemes, which can accommodate varying latencies, throughputs, client processing power, and server work loads. These schemes will optimize prefetching and refinement of multimedia, on the basis of semantic cues present in the streams and related to anticipated as well as past work patterns. They also corresponds to the components subjected to the continual validation.

In order to proceed with the case study, we will have to build an experimental continual validation infrastructure and also a proof of concept multimedia provision system, along the lines sketched above. The latter will also allow us to better explore some issues which are relevant to the primary goal of this proposal, such as point 2 above, which provides some options for dealing with WF data distribution aspects (see also Section 5.3).

4.1.1 Application level WF in AI2TV

The distributed WF aspect in AI2TV is essential, since on one side it enables and enforces dispersed teamwork, while on the other side represents the source of knowledge that is fed into the multimedia infrastructure to enable adaptivity and efficiency of the content provision.

The typology of this WF is in general that of a classic human-oriented process, in which WF agents represent persons and the goal of the process is to facilitate and guide the collaboration among those persons. The process is explicit, well defined and fairly open-ended. The most relevant WF distribution dimensions concern the enactment architecture, the state of the process – which must be fully available to the multimedia provision system – and run time access to artifact/resources (including multimedia).

Many other characteristics of the WF strictly depend on the application domain of the teamwork: in our case study we plan to address educational processes involving SW development, such as the case

of geographically dispersed student teams carrying out course project assignments, but other collaborative contexts involving multimedia streams can provide equally valid application domains.

In fact, the presence of streaming multimedia is a source of peculiarities for the WF. First of all, multimedia streams and fragments thereof provide an excellent example of information impromptu resources, since relevant fragments must be dynamically determined and extracted from the raw data mass. Moreover, they add a temporal concern, which is mostly overlooked in traditional WF, which regards the acquisition and the utilization of artifacts and resources as instantaneous with respect to task invocation. That is in all cases an approximation, whose validity limits do not encompass multimedia types of data in two respects: the non-negligible access latency in a truly distributed environment and the lower-bounded time necessary to operate a task that requires the assimilation by users of some information contained within a multimedia stream. While the former aspect can be handled as a specific facet of the more general WF data distribution problem, the latter is inherent in the nature of the multimedia data employed by the WF, and analogous considerations may arise for other non-instantaneous impromptu information resources, e.g., successively refined Web searches. An interesting open question is whether this outlines a temporal distribution dimension of WF. If so, should enactment engines account for non-instantaneous access and utilization of certain artifacts and resources, and how? And how should this be described in the process model?

Table 4 in Appendix A provides a synopsis for an application WF in the context of AI2TV.

4.1.2 Continual validation and AI2TV

In AI2TV, a continual validation meta-architecture (and the associated infrastructure level WFs) may be used to guide the behavior of the provision infrastructure. Specifically, it can enable the dynamic selection of the video management schemes and the transmission modalities (ranging from streaming of full content to prefetching/caching of selected segments), on the basis of conditions detected by probes placed onto the various components of the provision infrastructure, as well as the client environments and the networking interfaces on both sides.

The meta-architecture gauges capabilities, performance levels, problems and failures, synchronization states and so on, and reports this wealth of monitoring information to a set of controllers that collaborate to emit directives for the automatic adaptation of provision parameters. Figure 4, represents informally the interrelationships between AI2TV and the continual validation meta-architecture.

5 Mechanisms for enhanced distribution of WF

In this Section we describe a set of mechanisms that we believe can effectively enhance the way a distributed WFMS enacts WF, and which consider and help reconciling the various distribution dimensions discussed earlier on. Those mechanisms individually and collectively approximate the optimization of distributed WF enactment, particularly in terms of enhanced availability of data and resources to the WF agents. We are familiar with and consider exploiting those mechanisms for constructing experimental systems that investigate the coordination reference problems in the distributed WF perspective. Other candidate enabling technologies with different approaches but similar potential are surveyed in Section 6.3.

5.1 The dataweb approach to the distribution of semantically rich information

Datawebs [79] [80] are an effective way for organizing and structuring information according to its semantic model in a coherent and unified way, without imposing any restriction upon the location of any piece of data. Datawebs superimpose networks of unobtrusive hypertextual metadata upon a (potentially hugely) dispersed pool of data, enabling straightforward categorization, *structural browsing* and retrieval.

The term dataweb defines an information base and the corresponding semantic model. The information content of a dataweb is represented as a set of URL or URI-accessible resources, which can take the form of HTML pages, XML documents, images, binaries, etc. The semantic model explicates the intended structure of the information base: in the original implementation, the semantic model of choice is an E/R

schema [113], but other semantic models[2] can be employed and implemented upon the same principles. The semantic model is expressed with various layers of metadata upon the information base: metadata is represented as a set of Web pages separate from the information base. WWW hyperlinks connect opportunely the various layers of metadata with each other and to the information base. One layer provides descriptors for the resources taking part in the information base with hyperlinks pointing to their URL or URI; it also provides similar descriptors for any relationships, correspondences, dependencies, etc. holding among them according to the semantic model: the relationship descriptors also include hyperlinks to the descriptors of the correlated resources. The following layer provides directories of hyperlink pointers to the above-mentioned descriptors grouped by type (i.e. by each Entity or Relationship in the case of an E/R schema). The top layer describes the topology of the schema, with pointers to all the directories.

In Figure 6, a fragment of a dataweb taken from a SW engineering example is shown, to highlight the semantic layers superimposed on the information base and their structural interrelationships.

A dataweb constitutes a variation of the open hypermedia system (see Section 6.3 for more details) idea, with the peculiarity that data, metadata, link bases, types and semantic schemas are all stored referenced and accessible directly on the Web. Datawebs promote a structure-driven access and utilization of their content, but - in contrast to most OHSs - are very lightweight and completely Web-native. Furthermore, the level of scalability and decentralization of information supported is the same as that of the WWW.

We maintain that all the data distribution dimensions of WF can be gracefully handled by and expressed with a set of properly designed datawebs, provided compliance with the global networking scenario; that is, the dataweb concept must be extended, so that it is made available to WF components on network nodes that are not Web-based, through an adequate set of interfaces and bridges.

The *WF specification dataweb* can express both WF declaration and definition: it can organize and provide a unified view of all the distributed WF declaration fragments, correlating them with respect to their mutual dependencies, and also explicating any such relationships. Moreover, WF actions can be referenced using the same dataweb, with relationships connecting each WF fragment declaration to the corresponding action.

At any moment during the execution of the WF, various instances of the WF tasks are enacted and are in some specific state. This kind of WF information can be heavily decentralized, since it is owned by the distributed WF agents carrying out the tasks, i.e. those who have taken the responsibility to uptake a task and are executing the corresponding action. For on-line process monitoring and auditing purposes, however, it is convenient that information about enacted WF instances and their state is published by WF agents in an appropriate and structured form. Once again a (rather dynamic) *WF state dataweb* can be used to structure and reference all the dispersed enactment-time data. This dataweb can thus provide a unified view of the state of the overall process, and can also be easily linked to the WF specification dataweb, according to a type/instance semantic schema.

Distributed artifacts and resources needed by the WF, together with their types can be made available through a *WF document dataweb*. It can provide a unified representation of the types and the relationships holding among types, and can link such a type system to the actual instances of artifact and resources, according to a type/instance semantic schema. Thus, the WF document dataweb can effectively take the place of the artifact repository in a classic centralized WFMS.

As far as (information) impromptu resources are concerned, many times they are not part of the artifact repository from the start, and must be incorporated in it whenever they become available. To achieve this, it is possible to classify the (information) impromptu resources either within the same type system as the other artifact and resources (hence within the same dataweb), or keep them in a separated *impromptu information dataweb*, which might be organized according to a different semantic model (e.g. according to the task(s) they take part in, rather than the artifact/resource types). In the latter case, the two different datawebs can be overlapping, i.e. reference some common information in different ways, thus providing two different views on these categories of WF data.

We foresee that the dataweb approach can be useful to attack data-related WF distribution dimensions (see Section 2.2). It is noticeable that the overall amount of WF data needed at both design and run time can be all expressed and orderly distributed with a set of interrelated or overlapping datawebs: so far we have

[2] For instance, a dataweb devoted to describing the artifacts of a software development process could employ the semantic model of UML class diagrams [85].

mentioned a WF specification dataweb, a WF state dataweb, a WF document dataweb, an impromptu information dataweb. Each of them will provide a self-consistent and complete view about one of the data-related distribution dimensions without perturbing any of the others. Also, other views can be added as needed, by referencing and interconnecting in different ways (i.e. according to different schemas) the metadata descriptors in one or more of the mentioned datawebs, thus obtaining additional datawebs. In this context, XML and related formats and mechanisms can be used extensively for expressing and manipulating the metadata in forms that are amenable for both human and computerized agents.

5.2 Task forecasting

Task forecasting is the ability to predict the (likely) future course of an enacted WF, on the basis of some heuristics. It can have different degrees of sophistication. Many traditional WFMS are able to indicate what is the most likely (i.e. the default) WF course, at least at the time of initial enactment. Another simple forecasting technique is to "look forward" one step from the current enactment stage and pick either the most likely or all steps that can legally follow.

A more sophisticated kind of forecasting would be able to compute the most likely path from any given enactment stage, on the basis of knowledge about the current WF state, or even to rate the likeliness of any alternative courses. The ability to carry out this kind of forecasting strongly depends on the kind of WF formalism employed and on the level of detail and comprehensiveness of the WF state information. Moreover, in the case of hierarchically defined WF, forecasting may either be limited to one hierarchical level only (i.e. within the same subplan), or traverse part or all of the hierarchy of the WF.

Task forecasting can also greatly benefit from availability and analysis of the WF history. Effective predictions can be made not only and not necessarily by reasoning on the layout and the possible state transitions of the WF, but simply looking back at what had happened during past WF instantiations [110]. Of course, history-based task forecasting becomes increasingly effective only with time and after a sufficient number of repetitions of the WF, and its reliability is proportional to the repeatability of the process.

Independently from the techniques employed, task forecasting capabilities may enable the WF agents and infrastructure to prepare for future work. More specifically, they can be precious in guiding the adaptation of the distribution of the WF data or architecture [111] according to the forecasted work needs.

In our context, task forecasting capabilities must be available to the various decentralized WF agents. Therefore, even the forecasting "oracle" must be decentralized or at least replicated.

The scope of the predictions may vary depending on the type of WF; for example, in a hierarchical WF task forecasting for an agent may take into consideration possible paths of WF execution within the subplan including the WF fragment currently enacted by the agent,

We plan to associate task forecasting capabilities to smart caching (see Section 5.3), in order to enhance the availability of WF information to WF agents; we also mean to associate them to mobile agent technologies, in order to be able to deploy WF agents and assign them work in a dynamic and optimized way.

5.3 Smart caching mechanisms for distributed WF

Caching techniques are used to reduce average access latency: we focus here on documents shared among dispersed users or teams thereof, in the context of some decentralized collaborative process, in which network latency is likely to be an issue. Zero latency refers to the situation in which document access produces a cache hit on a (relatively) local machine, that is, there is negligible latency due to the network. This is the typical assumption made by traditional WFMS about artifact and resource access. Positive latency is the situation in which non-negligible latency delays the utilization of a requested. Finally, negative latency refers to automatic presentation (pushing) of a document to a user based on an accurate prediction that the user will need that document. It is important that the caching system is "smart", i.e. able to express and comply with very precise retrieval and push criteria, in order to avoid information noise, which might overwhelm users.

The proposed *Workgroup Cache* system [105] for smart caching can leverage, in principle, any knowledge available about the semantic content and pragmatic usage of documents as a basis for prediction of future accesses. The Workgroup Cache focuses the potential semantics and pragmatics with respect to a

"workgroup", that is, a set of users working on the same or related tasks. The criteria to instruct the Workgroup Cache are very flexible and can be derived from the workflow routing among workgroup members, as well as document access patterns of the workgroup members, or drawn from XML metadata associated with accessed documents. Criteria might be defined via simple filter rules, like Web search queries, as well as via a very elaborate event/data pattern notation.

5.3.1 Smart, team-oriented caching

A smart cache, such as the Workgroup Cache, is a useful mechanism for any widely distributed CSCW, in a number of ways:

- Negative-latency caching is a way to push to a work group or an individual (potentially) relevant information for their activities.
- Zero-latency caching is a way to benefit from past experience and data collected by the work group or individual.

When the criteria for smart caching are derived from the specification and state of an instantiated distributed WF - the above mentioned benefits can be extended to WF agents – either human or computerized - and can represent a solution to the problem of efficient dislocation of WF data/resources with respect to the WF agents. Specifically, negative-latency caching can be particularly effective for this purpose, when coupled with task forecasting capabilities.

Also the issue of the impromptu resources that must be collected for carrying out a task can be taken care of by appropriate negative-latency caching mechanisms. Such impromptu resources can be either data or the computations that must be run to produce some necessary WF data. In this respect, smart caching capabilities can be of further use if associated not only to (groups of) WF agents, but also to given WF tasks: the same impromptu resources, in fact, might be re-used for different instantiations of the same WF task, no matter what individual/group actually performs it from time to time. It makes therefore sense to associate a smart cache also to task definitions.

The Workgroup Cache can therefore be seen in this context as:

- a local pool that mirrors remote WF data and impromptu resources, for the benefit of some (group of) WF agents;
- an automatic mechanism to collect and maintain impromptu resources in a way that is transparent to the WF and the WFMS;
- an auxiliary agent that can be used by WF agents to carry out the “sideways” steps necessary to secure some impromptu resources that are going to be needed to carry out some task.

In our context, each WF agent shall be equipped with a smart cache; agents that at some point in the WF collaborate towards some ends can put in common their individual smart caches and their contents, and create at any moment in the process a common cache for their workgroup. The caching criteria for the smart caches are provided by the agent (or agents), which exploits its knowledge of the running WF and any available task forecasting capabilities.

5.3.2 Smart caching and datawebs

The smart cache must be aware of the organization and semantics of the WF data to be effective. With respect to the dataweb context, this means to be able to recognize, access and manipulate the existing WF-related datawebs, properly taking advantage of the superimposed layers of metadata. In fact, the availability of an explicit semantic model for the WF information, as that provided by datawebs, enables the smart cache to work in a focused and efficient way, minimizing both information noise AND silence.

However, the smart caching mechanisms must also be able to fetch and handle generic, unstructured information residing on the global network, which may be relevant to the WF (e.g. impromptu information resources). A possibility that is worth exploring is organizing of all the cache internal storage in the form of yet another “local” dataweb, which can be interconnected with other datawebs of cached information as well as with the “main” datawebs that take care of the wealth of original WF information.

As far as the implementation of the caching mechanism is concerned, it can have any degree of sophistication, ranging from a relatively simple and passive proxy, to a mobile information retrieval agent,

to a community of ancillary intelligent software agents, as far as the functionality and the interaction model made available to WF agents remain the same.

5.3.3 Smart caching: open issues

A cache contains only a work copy of the original information, while the master copy remaining on the original host. Various issues derive from this duplication, such as the resolution of concurrency conflicts among various cached copies, the refresh of the work copy and its granularity, etc. In a widely distributed and rather dynamic context such as that of decentralized WF enactment, these issues can become seriously problematic. Datawebs have built-in update and concurrency policies and mechanisms, which address the direct manipulation of the master copy on the remote host by multiple users. Those mechanisms must be extended and integrated with the concept and the functionality of the Workgroup Cache.

5.4 Dynamic deployment of WF agents

The dynamic dimension of the architectural distribution of WF may be supported in different ways and at different levels by a distributed WFMS. Hereby we list - according to an increasing level of dynamism – various approaches to the dynamic deployment of WF agents:

- support to mobility of human agents
- support to disconnected WF operation by human agents
- on-the-fly instantiation of static computerized agents
- support to the "cloning" of static computerized agents on remote hosts
- support to the migration of mobile computerized agents[3]
- support to sudden activation/deactivation of agents as a consequence of spontaneous networking.

Independently from the level of support to dynamic reconfiguration of the architecture offered by a given distributed WFMS, the execution of a decentralized WF would benefit from mechanisms that carry out such reconfiguration in a coordinated and timely manner, and furthermore as transparently as possible with respect to the ongoing WF enactment and its users. For instance, ideally users should not be forced to specify explicitly when and where to instantiate a new remote computerized agent, and for what purposes, but should rely on the WFMS timely recognition of situations in which this need exists.

Once more, task forecasting capabilities can play an important role, since they can also be used to *pre-deploy* WF enactment agents onto any network host that is suitable to the execution of a (series of) forecasted tasks, in case the current distribution of the WF enactment architecture is not convenient. Such an automatic preventive deployment of WF agents (or also re-deployment, in case the architecture of the WFMS supports some form of agent mobility) clearly regards computerized agents only.

An important open issue is that of the interaction between the data pre-fetching capabilities through smart caching and the preventive agent deployment capabilities in the same WFMS. In many cases, the pre-fetching and pre-deployment can be seen as alternative, e.g. either pre-fetch the necessary information to the location of an active WF agent, or pre-deploy a WF agent to the location of relevant data for the WF. An integrated view of these two capabilities together with the task forecasting capabilities must be conceived – perhaps through proper decision-making policies and heuristics - in order to maximize the effectiveness of pre-fetching coupled with pre-deployment.

Another issue regards the interaction between the dynamic relocation of WF computerized agents – with their WF enactment engines - and the dispatching of task declarations and definitions to be enacted by those agents. It is clearly possible to keep separate those two concerns: in this case, the computerized agents are pre-deployed onto the target host to provide the *enactment context* and the machinery for executing some WF tasks remotely, while all the necessary enactment directives are pushed at a distinct time and through different means. But it is also possible to pre-deploy the agent and the WF data together, thus explicitly binding together in advance the task to be enacted to the agent. In this case, the distinction between the distribution of the architecture and the distribution of work to the architecture is blurred. On the other hand, it fits especially well the idea that the architecture of a decentralized WFMS should be not

[3] Notice that SW mobility itself has various degrees, each with its distinct properties [75].

only dynamically reconfigurable, but also the reconfiguration of the architecture should be merely consequential with respect to the dynamics and the distribution of the WF enactment.

Both options can be explored to various degrees with a technological infrastructure for the computerized agents that supports mobility, coupled with some form of process awareness, like in the case of *Worklets* [106]. Worklets can aid to express and convey to the decentralized WFMS both directives for the dynamic reconfiguration of the enactment architecture, and directives for the distribution of the enactment itself. Alternative agent platforms, such as the Aglets [107], could also be profitably used in this context.

5.5 A conceptual architecture for advanced decentralized WF

The construction of a WF solution that handles the reference problems in the domain of distributed systems coordination and operates over all the distribution dimensions is an ambitious endeavor. A number of techniques that can enable (parts of) such a solution have been proposed earlier in this Section. They constitute a rather composite set of mechanisms, whose integrated implementation within the architecture of the WFMS must be studied carefully. Also, some of those mechanisms may result incompatible with each other, or be substituted by others that are functionally similar or equivalent. At this stage, we present a conceptual architecture, based on considerations about WF distribution and assumptions about the roles that some of the mechanisms above can play, and descending primarily from principles that are briefly outlined below

We envision a communication facility that ties together all the many components of the WF architecture over a global networking environment: it is some kind of *Global Bus* for efficient and dynamic routing of the multiple data and control flows exchanged within the architecture, whose formats and semantics may be extremely diverse, but whose circulation should be handled in a unified way, notwithstanding the heterogeneity of the global network. Several issues regarding the Global Bus are fundamental for the design of a decentralized WFMS; for this reason, we devote to it a brief separate discussion in Section 0.

WF agents must be treated uniformly, independently from their nature (operating interfaces for human stakeholders, or wrappers of legacy systems and tools that are incorporated in the WF, or largely autonomous computerized agents) and their purposes. This is a generality requisite, to allow the WF to handle effectively a wide variety of processes, e.g. cooperation among humans as well as coordination of automated distributed systems. Therefore, all agent environments must be equipped with the same functionality and must have the same access to WF information. An agent environment can be seen as a collection of SW components offering a set of capabilities enabling WF enactment in autonomy as well as in collaboration with other agents. No pre-defined hierarchy between WF agents exists, in the sense of centralized WF enactment decisions.

The WF architecture can also include “ancillary” SW agents that are not primarily concerned with the enactment of the WF, but offer additional capabilities that can be useful or even essential for the overall decentralized WFMS (some examples can be the decentralized oracle for task forecasting, or information gathering agents for delivering impromptu information resources,). Deployment and utilization of those ancillary agents occurs as a consequence of “decisions” taken by the WF agents.

In Figure 5, an informal diagram of the conceptual architecture is drawn, which, although approximate and incomplete, tries to highlight the multiplicity of components taking part in the foreseen WF solution, the diversity of data and control flows, and the complex network of relationships between all the elements.

5.6 The Global bus communication infrastructure

In the first place, the Global Bus is a useful abstraction that enables uniform reasoning about all the inter-process communications, independently of any machinery, protocols, semantics, etc. Besides, a Global Bus is also a concrete architectural device, whose design and implementation must provide a potentially extremely dispersed but conceptually and functionally unified communication facility of great versatility. The Global Bus may be seen as a sort of middleware (see Section 6.3 for details) for the decentralized WFMS; however, it must be noticed that it must have some peculiar characteristics.

First of all it will service a decentralized WFMS, which shall be active along the dynamic architectural distribution dimension of WF; thus WF agents could migrate on potentially any network node on very platforms, and the Global Bus will need to be able to gracefully adapt in order to remain available to the

migrated agents. Therefore it must not be tied to any specific inter-process communication paradigm but must cross boundaries among them, coherently with what stated about a global networking environment in Section 2.1. An option we intend to explore is a sort of XML-based middleware (as advocated for example in [86]), which would use some form of encoding of all inter-processes communications with XML DTDs, and will use the exchange of XML streams as the common denominator across platforms.

Moreover, the Global Bus must support a great deal of dynamism, mobility and on-the-fly reconfiguration of its own architecture. As an infrastructure, it is particularly affected by changes in the topology of the underlying network it provides an abstraction for, such as those consequential to spontaneous or active networking: the Global Bus must be able to continue its services notwithstanding the volatility of the network medium, and also to offer them as soon as new spontaneous connections are established between new nodes, on whatever medium happens to accommodate elements of relevance for the decentralized WF. The deployment structure of the Global Bus is bound to be as volatile as the network, and as dynamic as the active components that exploit it to communicate. Thus, the Global Bus must have mechanisms enabling fluid and seamless transitions between configurations, and must ensure reliability and availability with respect to and for the components of the distributed WFMS during and after all those transitions.

6 Related work

The proposed research crosses over and draws from a number of domains. In the remainder of this Section, we try to provide an overview of the state of the art in those domains, at the same time highlighting the intended contributions of our research with respect to them. First of all, we describe some of the latest examples of decentralized WF technology, pointing out the WF distribution dimensions they cover. Then, we offer a view over the very wide and composite field devoted to the study of coordination models and languages, addressing general-purpose contributions, as well as results from other specific Computer Science domains concerned with coordination issues, and we will try to assess the potential of distributed WF technology for coordination with respect to those other approaches. We conclude by analyzing some enabling technologies and their relevance for our purposes, such as open hypermedia systems (correlated with datawebs), push technologies (correlated with smart caching mechanisms), and middleware frameworks, which can be seen at the same time as instrumental and alternative to other approaches for the development of distributed SW.

6.1 Distributed WFMSs

We describe hereby some results in decentralized WF technology that are meant to operate at a very large scale of distribution, typically the Internet scale, which also represents the reference domain for our studies. In Appendix B, we provide a synopsis in Table 5 for a number of the WFMSs discussed below, profiling them against to the identified distribution dimensions.

Endeavors [20] is both a research prototype and a commercial WFMS, strongly based on WWW paradigms and protocols. It uses HTTP as its main distribution and communication mechanism, complemented and extended by means of Servlets [28], and WebDAV [29]. WF descriptions are coded in XML and accessed via the SWAP protocol [30] . WF descriptions are hierarchical, with increasingly more refined *activity networks*, which are sets of activities associated by control, data and resource flow. The leaves of the hierarchy are atomic activities. WF is specified in an OO fashion in terms of artifacts, activity networks and resources, which are collectively termed Process Objects (POs). POs are stored persistently in object stores sitting behind HTTP servers and deployed according to different architectural schemes. The most general and flexible architecture is multi-client and multi-server [31], designed for scalability and robustness: POs can either live on the client environments (i.e., copies of Endeavors allocated to stakeholders), or on any of multiple HTTP servers, which are enriched with machinery to handle POs, which effectively constitute computerized WF agents. All POs communicate asynchronously by means of events that are routed through the HTTP Servers. In response to the events, POs invoke and execute matching *handlers*. Handlers are snippets of code that provide POs with behavior and can be dynamically bound to POs, possibly by transferring them on-the fly from remote object stores. Caching schemes for the WF information such as POs and handlers is supported. Concerning the distribution of work, POs representing activity networks are loaded onto the client environment either by pulling their URL explicitly on the user's part, or by pushing, via a delegation mechanism called the WebNavigator.

OPSS is a prototype of a distributed WFMS employed to coordinate Value Added Services within the ORCHESTRA telecommunication infrastructure [32]: services are a combination of distributed SW (often OTS) and human operations. WF in OPSS is specified in terms of Agents, Activities, Artifacts and Resources; each of these elements is characterized by a Finite State Machine (FSM) describing its state, as well as the valid transitions among states and the conditions that must hold for the transitions to take place. The aforementioned elements are organized in an OO fashion, therefore support specialization, including that of their characterizing FSMs, and are implemented as Java classes. The WF description is made up of one or more chains of activities, whose behavior (the associated *activity description* – a snippet of procedural code) is accessible via an URL. In OPSS, there are three kinds of WF agents: SW agents are completely automated and are proactively activated to enact some WF fragment; human agents are represented by an agenda-like interface for WF stakeholders who must carry out creative, open-ended tasks in the WF; external tool agents are wrappers needed to invoke any legacy or OTS SW exploited in the WF for business-specific tasks. Human agents execute a WF by pulling or pushing (delegating) activities to other agents. State transitions during the life cycle of activities fire events that may initiate other activities, which are assigned automatically to some agent. In its current implementation, OPSS include a centralized *State Server* maintaining the States of all elements in the WF, while WF agents, activity execution and artifacts can be distributed everywhere on the WWW. WF agents communicate among themselves and with the State Server via asynchronous events.

Serendipity-II [21] is a decentralized WFMderived from a more centralized predecessor called Serendipity. Serendipity-II is purely component-based: each Serendipity-II agent is in itself a set of interconnected components. Also all WF information (models, states etc.) is expressed in the form of software components. Serendipity-II stresses collaborative distributed process editing – as well as enactment - and supports disconnected work. The process description is a hierarchy of *process stages*. Process stages have a state, are assigned to *roles* and are interrelated by enactment flows, which are traversed when the corresponding state for a process stage is reached. WF agents in Serendipity-II represent WF stakeholders (humans), as well as computerized agents, and interfaces to external tools. Each agent comprises a complete copy of the WF modeling and enactment environment. Each environment executes a (part of the) WF specification autonomously and communicates with other agents via events called *change descriptions*. Users enact process stages by pulling the specification of some stage, and the work flows both in the local environment and in any affected remote environments, according to the events fired when enacted stages change their state. Component-passing/copying schemes, including the circulation of change descriptions and their logging for momentarily disconnected agents, allow the exchange and versioning of all WF information that must be shared among WF agents: at the cost of partial duplication, there is no centralized server or repository for WF information. The architecture also contemplates ancillary, fully automatic software agents, called *work coordination agents*. These agents match and filter the events exchanged by WF agents and execute processing actions in response to filtered events. Actions are fully customizable depending on the application, and include the automation of chains of WF tasks, WF logging, legacy tool integration via invocation of wrappers, etc. Work coordination agents can be deployed at the users' will.

$METEOR_2$ [35] is a *model* for distributed WFMS. There are various implementations of the $METEOR_2$ model, such as OrbWork [34] (based upon CORBA middleware and services), and WebWork [33], which relies solely on WWW technologies. All implementations share the same formalism and facilities for process modeling, and differ with respect to the enactment infrastructure and environment. Hereby, we examine the lightweight WebWork incarnation of $METEOR_2$. WF is described as a hierarchy of tasks, with tasks at the same hierarchical level interconnected via inter-task dependencies, which define both control and data flow. Each task has its own FSM, which defines valid states and transitions. Transitions to a given state can also be forced by dependencies from external tasks. WebWork is a multi-server, WWW-based WFMS. WF servers for WebWork are WWW servers; each WF task is allocated firmly to a specific WWW server and is executed on that server Servers accept and execute requests expressed via CGI from the users' browsers, potentially with some rerouting. The scheduling of the tasks is carried out by WF schedulers, which – although conceptually autonomous entities in the architecture - are in practice encapsulated into the WWW servers that execute the tasks, and can automate chains of tasks depending on the state of interconnected tasks.

Opera [27] defines a kernel that can provide WF services to a generic distributed system. As the term kernel implies, it is not meant to be in itself a complete WFMS, but a platform that needs to be extended and tailored to the application domain.

Oz [89] addresses WF distribution by pioneering the concept of WF federations. A federated WFMS [22] allows the sharing of WF fragments defined within the processes of diverse and dispersed organizations. Architecturally, Oz extends the client/server paradigm with a multi-server approach.

APEL [98] [99] embraces the idea of federations to integrate heterogeneous WFMSs, each of which serves as a component of a larger-scale WFMS, with a distinct functionality. The various WF representations and semantics, as well as the various levels of operation, are reconciled by a *Common Process Engine*, enacting a common WF model, which represent partial centralization points for the overall WF, whose degree of decentralization is for the rest largely influenced by that of the WFMSs participating in the federation.

Juliette [100] – based on the Little-JIL [101] process modeling formalism – presents a hierarchically distributed and mobile WF engine (the *interpreter*), and a system of distributed agendas to keep track of the WF state. The WF distribution is eminently proactive, since the interpreter finds through a *resource manager* an agent capable to enact a WF step and then dispatches to its agenda a corresponding work item.

IBM MQSeries Workflow [102] is one of the leading industrial WF products. It basically adopts a variation of the client/server model, with Web-enabled clients and multiple servers organized in a hierarchy. Also, to optimize local workload, servers at the same node in the hierarchy tree can be distributed on separate machine. The WF specification is inherited according to the hierarchy and can be specialized locally.

Mobile [87] and Exotica/FMQM [88] adopt a distribution approach based on the replication of WF servers – which however remain points of centralization - with complex schemes for the preservations of consistency and for work partitioning and assignment among the servers.

6.2 Coordination languages and models

The study of coordination has lately gotten a lot of attention in a variety of scientific disciplines, besides Computer Science, such as Operations Research, Organization theory, Economics, etc. An accepted interdisciplinary definition of coordination, which does not constrain in any way the nature of the coordination subject, but emphasizes the *process* underlying coordination, has been proposed in [1] : "*The management of dependencies between activities*".

Within Computer Science, a number of more specialized definitions have been proposed, each reflecting different perspectives on coordination problems, such as SW architecture specification, component-based SW frameworks, Distributed Artificial Intelligence, agent-based systems, WF, and others. Each of those perspectives studies coordination models, languages and mechanisms with different purposes and scopes, although of course the main motivation is common, i.e., the specification and guidance of the interactions among entities taking part with different titles to a complex SW system.

A large quantity of effort has been also devoted to studying general-purpose coordination concepts that can be employed across application domains like the ones listed above. This trend was perhaps initiated by Carriero and Gelernter, who proposed in [2] the strict separation of concerns between coordination and computation in programming. Such separation should provide a clearer understanding of the complexities inherent in guiding the interactions within a distributed system, and a key to deal with generality and heterogeneity concerns (e.g. cross-platform, cross-language etc.)

According to [2] , coordination is the process of building programs by gluing together *ensembles* of active entities; a coordination model takes the role of the glue that binds together the computational activities carried out by the entities in the ensemble, and a coordination language is the linguistic embodiment of a coordination model, offering facilities to express synchronization, communication, creation and termination of the coordinated computations.

A seminal example of a pure coordination language is Linda [3] [4] , which provides simple but powerful linguistic means and architectural abstractions for the coordination of generic distributed systems, as well as for computational parallelism. Linda's coordination model is based on the concept of a *tuple space*, i.e. a global shared data structure that serves as the only mediator of the interactions among all components of the system. The tuple space model owes much to the classic blackboard architecture [95] [96] [97] of many Distributed Artificial Intelligence (DAI) systems.

Although implicit, since it is completely data-driven [5] , the coordination model promoted by Linda can be implemented easily on top of most conventional programming languages [6] and its quite general. This is why Linda has become a reference point for new coordination models and languages, and the reason behind its numerous variations, derivations, and specializations in a myriad of Linda-based models and systems,

which address a variety of application domains (an example out of many: JavaSpaces™ [7] by SUN Microsystems).

Many other general-purpose languages, sporting different paradigms (e.g. control-driven[5] langauges) have been developed and studied in the last few years (see for example [5] and the COORDINATION conference series [8] [9] [10). However, it is also interesting to discuss some results originating from fields of study with a more restricted scope, but which provide nevertheless useful insights into the Computer Science coordination problem space at large.

Architecture Description Languages

The study of SW architectures, for example, has produced a number of Architecture Description Languages (ADLs) [11], whose purpose is the high-level but to same degree formal specification of the structure and behavior of a SW system. ADLs predicate about *components*, *connectors* and *configurations* (i.e. the topologies of component and connectors instantiating an architecture). As observed in [12], all inter-component interactions are captured by connectors: different types of connectors support different coordination models, and, once instantiated in a given configuration, determine its valid coordination models. Although many of the latest ADLs provide a good deal of tool support to the guidance of the lower phases of the SW development process, aiming at enforcing the architectural decisions onto the low-level design and also the implementation phases, ADLs remain prominently declarative.

Coordination programming

More imperative connotations are present in Module Interconnection Languages (MILs) [14] 15] and *megaprogramming* paradigms [13]. Both promote a bottom-up, *compositional* approach to coordination, which addresses the implementation of distributed systems by gluing together in a programmatic way already coded components with varying degrees of autonomy, heterogeneity and granularity. Only megaprogramming is – as today - an active thread of research: megaprograms operate at very coarse granularity, and aim at the construction of *metasystems*, by encapsulating and composing megamodules (i.e. services which can constitute large, complex, coherent and complete distributed SW systems in themselves) across their different *ontologies* [16] . In theory, megaprogramming languages are only concerned with expressing and scheduling computation requests to the various megamodules according to their ontologies, and to return results to other megamodules, applyinginter-ontology correspondences when needed (which are conceptually challenging). They must remain independent from and unconcerned with the implementation technicalities, the computing and communication infrastructures, and the application domains proper of the megamodules, which is in practice a considerably difficult technical challenge.

Coordination of active agents

Many coordination languages, and certainly ADLs and megaprogramming languages, adhere to a conventional view in which components are “passive” subjects of coordination. Nowadays, that view can be too restrictive: SW components can be “smart” and “active” agents, which sport characteristics such as substantial autonomy, awareness and knowledge of the application domain, some degree of reasoning and decisional power, mobility [83]. SW agents can be part of self-organizing communities, causing the coordination model to be dynamically influenced by the very subjects of coordination [81]. The coordination of agent-based systems thus requires a different paradigm with respect to that of traditionally engineered component-based SW.

Agent-based systems are rooted in DAI, but their experimentation and usage has grown steadily in the latest years, principally due to the affirmation of the WWW as the dominant information as well as computational global infrastructure [109]. In that context, agents have been applied for widely distributed information gathering and processing, data mining, document management, electronic commerce, etc. A number of coordination models and the corresponding Agent Coordination Languages (ACLs) [82] –such as KQML [84] - have been explored in response to these trends. Most models are based on the concepts of *goals*, and *plans* that must be executed to reach goals. Plans are analogous to well-defined, explicit, often hierarchically decomposed processes. The level of flexibility and dynamism with which agents interpret and execute a plan depends on the chosen coordination model. It may vary from rather inflexible *organizational structuring* (i.e., a coordination policy defined a priori and pursued by a master coordinator of “slave” agents) [92], to fully dynamic run time *negotiation* [93], with agents interactively searching for

and pursuing the best compromise among each other's goals, exchanging information about plan fragments, services/capabilities and ontological contexts.

WF and distributed systems coordination

The primary goal of WF technology is the coordination of human-oriented processes, i.e. processes in which the subjects of coordination are persons, collaborating towards some common accomplishment. Such processes can be very diverse, are usually very open-ended with possibly long duration, and have a high degree of variability, calling for the handling of exceptions and multiple alternative courses of actions. Distributed WF faces many challenges, some of which have been discussed earlier in this document, which require them to rely more and more onto legacy, OTS, third-party or otherwise autonomous SW for the remote enactment of substantial WF fragments. Also in this case, the WWW has served as a catalyst as made evident by the big focus on WWW issues on the part of distributed WF research and industry.

By observing the efforts and trend in all the various fields mentioned above, it is possible to perceive substantial overlapping, the opportunity of synergy, and also hints of convergence. This is being increasingly recognized by scholars and practitioners (see for example [17] [23] [26] . In the latter: "*The last decade gas seen the emergence of a class of models and languages variously termed coordination languages, configuration languages, architectural description languages, and agent-oriented programming languages. These formalisms provide a clean separation between individual software components and their interaction within the overall software organization*."). In this convergence scenario, distributed WF has the chance to play a prominent role, since it has the potential to address many concerns typical of the other domains.

For example, substantial commonality has been detected between software agents and distributed WF [17] : on the one hand, it is feasible to try to orchestrate the cooperation of an agent community as a distributed WF, as experimented for instance in [18]; on the other hand, it is possible to employ SW agents to represent (some) WF actors or enhance WFMS functionality (as in [20] [21]), and to use ACLs to describe (some) human-oriented processes (as in [19] among others). Moreover, federated WFMSs [22] [99] represented an early although specalized case of coarse-grained componentware, in many ways analogous to metasystems, as in the megaprogramming paradigm. Megamodules are in that case whole WFMSs or components thereof, which are coordinated by means of federated processes to create a meta-WFMS. It seems feasible to extend the experience of federations from WF-dedicated components to generic SW components, employing WF specifications as their programmatic glue: [24] [25] –among others - advocate such an approach as a most important trend for WF research, and also consider the investigation of the interplay between ADLs at the specification level and WF at the implementation level of the distributed and federated system.

There are a few important reasons why distributed WF can be the "common ground" for the convergence of many coordination paradigms such as agent systems, megaprogramming and SW architectures. The notion of process is native and allows to describe coordination explicitly and abstractly at the same time. Moreover, most WF formalisms have both declarative and imperative connotations (in the terms used in this document, they map respectively to the declaration of tasks vs. the definition of behavior/actions attached to those tasks). Thus they are feasible for reasoning about the model of coordination (like ADLs), as well as implementing it over the distributed system (like megaprogramming languages). Finally, and opportunistically, WF technology is perhaps not mature *tout court*, but seemingly more than agents and ADLs, and certainly more than megaprogramming: for example, distributed WF accounts reasonably well for heterogeneity (of actors and technologies) and generality (of kinds of processes supported).

All of the considerations above contribute to a vision in which a WFMS could be placed at the core of a distributed system, serving as the coordination medium and perhaps taking the role of a dynamic and process-aware middleware [27]. This is in accord with the goal of this proposal, i.e. the investigation of the potential of such a coordination perspective, in the context of the current state of and upcoming advancements in WF research and technology.

6.3 Enabling technologies

6.3.1 Open hypermedia systems (OHSs)

Open Hypermedia Systems (see [36], [43], and the series of Proceedings of the Workshop on Open Hypermedia Systems [37] [38] [39] [40] [41]) are concerned with the clear separation of contents (i.e., documents) and structure (expressed as hperlink sets) in hypertexts. OHSs like Chimera [42], Hyperform [46] , or Multicard [47], provide link bases (or link servers) that are distinct from the document repositories, superimposing dynamically the relevant hyperlinks on documents requested by their users and displaying them on dedicated or extended hypertext viewers. OHSs also provide type systems for documents and links.

OHSs hence promote an alternative way of structuring the information and exploiting the hypertext concepts with respect to the WWW, in which links are embedded into documents, and the overall structure remains completely implicit. In the last few years, the OHS community has actively researched ways to integrate its efforts with the WWW paradigm as seamlessly as possible (see for example [44] [45]).

Link servers in OHSs account for enhanced flexibility and dynamism, for instance enabling easier automatic processing of links, and on-the-fly creation/adjustment of structured hypertexts. Xanth [48] , for example, exploits the distinct link information to activate and guide a series of pre- and post-processing modules in order to provide enhanced services (such as WF, presentation, transactions, versioning, etc.), whenever some document is requested by the user. The price to pay with respect to the embedded hyperlink paradigm is generally that of limited scalability of the OHS, in terms of both volume and dispersion. Access to a hypertextual document is in general heavier since it include queries to multiple repositories and processing of the partial results to obtain the final layout of the document. Maintaining the link base consistent is another complex issue. Moreover, link services represent a point of centralization and potentially of failure for the distribution of information.

6.3.2 Caching and push technologies

Prefetching of data into a cache, in order to optimize the performance of a system is a theme found in many domains in Computer Science. We concentrate hereby on results applied to distributed information systems and the WWW in particular. We also mention some related results in recommendation systems.

Various large-scale cache systems (usually hierarchical) have been developed for the WWW, such as Harvest [49], in order to let (groups of) users obtain documents from an optimal secondary source (the cache), rather than from its sub-optimal primary source (i.e., the original WWW server hosting the document). [50] enunciates three main principles for the design of large-scale distributed cache, i.e., "*minimize the number of hops to locate and access data on both hits and misses, share data among many users and scale to many caches, and cache data close to clients.*" and proposes a hierarchical architecture that exploits metadata about the location of information to achieve improved response time. The resulting prototype, named Cuttlefish, also uses pushing schemes to move documents to caches in the hierarchy closest to clients that are likely to use them, but solely on the basis of past usage patterns.

Among systems that employ prefetching, there are some that are intended for disconnected Web browsing: they found their prefetching policies upon explicit user's indications [51] , or also supplement them with past usage heuristics [52]. These are different from the Workgroup cache, since the latter is also based on criteria that are automatically derived from the work context of teams as well as individuals, e.g. the state of an enacted WF. The concept of process-based prefetching is explored in Laputa [53], which is specifically intended to enable disconnected software development within a collaborative Process-centered Software Engineering Environment and was limited to supporting an individual user.

Prefetching and pushing schemes are also employed to recommend to users potentially relevant documents in context. Most of them concentrate on individual users and their preferences or habits: Remembrance Agent [70], Fab [72] , and Ant World a[73], and many others all have their own means to "guess" what the context of interest is and what other information should be presented. Examples of recommendation systems that are tuned to the needs and interests of groups are the Knowledge Pump [71] and ReferralWeb [74], but also a plethora of proprietary systems on e-business Web sites, such as Amazon.com and others,

which match the buying habits of their customers to cluster them in “communities” and to provide them with potentially interesting purchasing advice and offers.

6.3.3 Middleware and component-based SW infrastructures

Middleware technologies [68], such as DCOM/COM+ [66] [67] [68], CORBA [63], Java RMI [65], DCE [64] , JavaSpaces [7], TIB/Active Enterprises [103] and many others take a pragmatic approach to the development of distributed SW systems, by providing a layer of software that abstracts out low-level networking details and offers “in a package” a uniform set of inter-process communication facilities and a common API for gluing distributed components. Each middleware infrastructure carry with itself some definite assumptions on the valid interaction modality among components, hence it leans towards certain architectural styles and constrains the variety of coordination models that can be implemented on top of it [61]. By paying this price in terms of generality, developers gain with the adoption of a given middleware certain productivity advantages, like the hiding of technicalities and and an established structural and semantic ground.

Nowadays, middleware products tend to expand their scope by offering additional services (see for example [62] for CORBA), which either provide different alternative or complementary communication facilities on top of the basic infrastructure, or augment it with other packages of high-level functionality for the connected components (e.g. transactions, security, mobility support, etc.).

Distributed WF in a middleware-based view can be seen as another additional service; an alternative persepctive is that of *ad hoc* middleware that is specialized towards servicing needs that are peculiar to distributed WFMSs [69]. A more radical approach, that is akin to the line of research of this proposal, is that of having WF placed within the kernel of the infrastrucure, as advocated in [54] , thus creating a sort of process-aware kind of middleware that can be exploited for the construction of distributed systems of any kind, rather than only WFMSs.

7 Evaluation

Among the intended results of this work, there is an evaluation of the potential of distributed WF to address the coordination of distributed SW systems, in the first place in the application domains represented by the two reference coordination problems, and, by appropriate abstraction of the lessons learnt, in generalized cases. State-of-the-art distributed WF technology will be evaluated along the distribution dimensions we have defined, to highlight capabilities as well as limitations with respect to the problems at hand; furthermore, extensions and improvements to the state-of-the-art, like (but not necessarily limited to) those outlined in Section 5, will be subject to experimentation and similarly evaluated.

The problem of how to carry out a complex evaluation of what is in substance an innovative technology , in terms of principles, criteria and techniques, must be placed in the general framework of Empirical Software Engineering. While the Software Engineering community has recognized that proposed improvements to methodologies technologies and practices must be scrutinized to weigh their actual – rather than claimed – impact on organizations, processes and products, no common consensus has yet been reached on the general principles, measures and protocols for such quantitative evaluation. As stated in [90]: *“At this moment in time, there is no widely held collective agreed model of the definition and role of empirical software Engineering (ESE).”*. While approaches to establish quantitative confrontations of Software Engineering results are available in some specific contexts (for instance, computing performance, or models for software cost estimation [91]) in other contexts and - more importantly - at the general level this is not the case.

The intended approach for this work is to point out a set of software quality factors, which are likely to be affected by the selection of a coordination paradigm over another for developing complex SW systems, and try to consider the effects that WF-based coordination has on them. An additional difficulty in our case may be that it can be difficult to establish some benchmark to measure advantages and disadvantages of our approach with respect to any chosen quality factors, especially since the intended application domain (i.e., continual validation) is in itself an innovative area. The possibility of employing as an alternative or additional case study an application domain that presents substantial challenges with respect to distributed coordination, for the sake of the evaluation, will be considered.

At this stage, it is already possible to outline that the results of this works should have an effect on the following quality factors of the distributed SW:

- Productivity
- Reusability
- Maintainability (corrective, adaptive and perfective)
- Performance
- Reliability
- Robustness

With respect to each of those factors, we will take in considerations a number of metrics that will be collected during the development of any case study; metrics shall be quantitative whenever possible. Results will be discussed in comparison to the state-of-the-art and the available literature.

8 Conclusions

A number of research issues will be explored by the proposed work, and we expect the emergence of interesting results in the following areas:

- evaluation of the state of the art in decentralized WF technology with respect to the various distribution dimensions outlined in this proposal;
- investigation of a number of conceptual and technological challenges for advanced WF technology, such as:
    - impromptu resources;
    - manipulation of multimedia artifact/resources in distributed WF;
    - temporal concerns in distributed WF;
    - WF distribution over active and spontaneous networks
- definition of the requirements of advanced, fully decentralized WF technology to further and completely address the distribution dimensions and the challenges above;
- exploration, design and implementation of mechanisms to support full distribution of a WFMS, according to the afore mentioned requirements;
- evaluation of the potential of fully decentralized WF for distributed SW systems coordination, with special attention to the following coordination problems:
    - run-time control of a distributed system
    - dynamic coordination of distributed cooperative computations
- presentation of a technological solution that addresses the coordination problems mentioned above in the scenario of continual validation;
- application of the solution to one or more case studies.

In Figure 7, a tentative, high-level work plan for meeting the goals above is presented in the form of a GANTT chart. We foresee a prototype-based approach, with two major rounds of study, development, experimentation and evaluation of results.

9 References

[1] T. W. Malone and K. Crowston, *The Interdisciplinary Study of Coordination*, ACM Computing Surveys, Vol. 26, no. 1, pages 87-119, March 1994

[2] N. Carriero and D. Gelernter, *Coordination Languages and their Significance*, Communications of the ACM, Vol. 35, no. 2, pages 97-107, February 1992

[3] D. Gelernter, *Generative Communication in Linda*, ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1, pp. 80-112, Jan. 1985.

[4] N. Carriero and D. Gelernter, *Linda in Context*, Communications of the ACM, Vol. 32, no. 4, April 1989.

[5] G. A. Papadopolous and F. Arbab, *Coordination Models and Languages*, Advances in Computers, Vol. 48, Academic-Press, 1998.

[6] S. Ahuja, N. Carriero and D. Gelernter *Linda and Friends*, IEEE Computer 19(8): 26-34, 1986.

[7] Sun Microsystems, *JavaSpaces Specification*, Technical Report, Sun Microsystems Inc., July 1998.

[8] *Proceedings of the 1st International Conference on Coordination Models and Languages (COORDINATION 96)*, Cesena, Italy, April 1996

[9] *Proceedings of the 2nd International Conference on Coordination Languages and Models (COORDINATION 97)*, Berlin, Germany, September 1-3, 1997.

[10] *Proceedings of the 3rd International Conference on Coordination Languages and Models (COORDINATION 99)*, Amsterdam, The Netherlands, April 26-28, 1999.

[11] N Medvidovic and R. N. Taylor, *A Classification and Comparison Framework for Software Architecture Description Languages*, IEEE Transactions on Software Engineering, 26(1), January 2000.

[12] M. Shaw, *Procedure Calls are the Assembly Language of Software Interconnections: Connectors Deserve First-Class Status*. Workshop on Studies of Software Design, 1993.

[13] G. Wiederhold, P. Wegner, S. Ceri, *Toward Megaprogramming: a paradigm for Component-Based Programming*, in Communications of the ACM, 35(11): 89-99, November 1992.

[14] R. Prieto-Diaz and J. M. Neighbors, *Module Interconnection Languages*. Journal of Systems and Software, 6(4): 307-334, November 1986.

[15] F. DeRemer and H. H. Kron, Programming-in-the-large versus Programming-in-the-small, IEEE Transactions on Software Engineering, 2(2): 80-86, June 1976.

[16] T.R. Gruber, *The Role of Common Ontology in Achieving Sharable, reusable Knowledge Bases*, in Principles of Knowledge representation and Reasoning, Morgan-Kaufmann, San Mateo CA., 1991.

[17] M. L. Griss, Q. Chen, L. J. Osterweil, G. A. Bolcer, R. R. Kessler, *Agents and Workflow – An Intimate Connection or Just Friends?*, Panel report in Proceedings of the Technology of Object-Oriented Languages and Systems USA Conference (TOOLS-USA 99), Santa Barbara, USA, July 30- August 3, 1999.

[18] David Jensen, Yulin Dong, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, Stanley M. Sutton, Jr., and Alexander Wise, *Coordinating Agent Activities in Knowledge Discovery Processes*, in Proceedings of Work Activities Coordination and Collaboration Conference (WACC 99), pages 137-146, San Francisco, CA, February 22 1999.

[19] M. Banville, *Sonia: an Adaptation of Linda for Coordination of Activities in Organizations*, in Proceedings of the 1st International Conference on Coordination Models and Languages (COORDINATION 96), Cesena, Italy, April 1996.

[20] G. Bolcer and R. Taylor, *Endeavors: a Process System Integration Infrastructure*, in Proceedings of the 4th International Conference on Software Process (ICSP4), Brighton, U.K., December 2-6, 1996.

[21] J. Grundy, M. Apperley, J. Hosking, W. Mugridge, *A Decentralized Architecture for Software Process Modeling and Enactment*, IEEE Internet Computing: Special Issue on Software Engineering via the Internet, 2(5): 53-62, September/October 1998.

[22] I. Z. Ben-Shaul and G. E. Kaiser, *Federating Process-Centered Environments: the Oz Experience*, the Journal of Automated Software Engineering, vol.5, no. 1, January 1998.

[23] G. Alonso, *Workflow. Assessment and perspective*, invited talk at the International Process Technology Workshop, http://www-adele.imag.fr/IPTW/IPTW/Papers/Galonso.prn, Villard de Lans, France, September 1-3, 1999.

[24] J. Estublier, *Is a Process Formalism an Architecture Description Language?*, in Proceedings of the International Process Technology Workshop, Villard de Lans - Grenoble (France), September 1-3 1999.

[25] T. Bolusset, F. Oquendo, H. Verjus, *Software Component-based federations architectures are software architectures too*, in Proceedings of the International Process Technology Workshop, Villard de Lans - Grenoble (France), September 1-3 1999.

[26] P. Ciancarini and A. L. Wolf, *Foreword to the Proceedings of the 3rd International Conference on Coordination Languages and Models (COORDINATION 99)*, Amsterdam, The Netherlands, April 26-28, 1999.

[27] G. Alonso, *OPERA: A design and programming paradigm for heterogeneous distributed applications*, in Proceedings of the International Process Technology Workshop, Villard de Lans - Grenoble (France), September 1-3 1999.

[28] Sun Microsystems, *The Java Servlet API Specification*, Technical Report, Sun Microsystems Inc., 1997.

[29] Y. Goland, E. Whitehead, A. Faizi, S. Carter and D. Jensen, *HTTP Extensions for Distributed Authoring – WEBDAV*, RFC 2518, Standards Track, Proposed Standard, February 1999.

[30] G. Bolcer and G. Kaiser, *SWAP: Leveraging the Web to Manage Workflow*, IEEE Internet Computing, 3(1):85-88, Jan-Feb 1999.

[31] P. Kammer G. Bolcer, R. Taylor and A. Hitomi, *Supporting Distributed Workflow Using HTTP*, International Conference on Software Process (ICSP5), June, 14-17, 1998, Lisle, IL USA

[32] G. Cugola, E. Di Nitto, A. Fuggetta, *Exploiting an Event-based Infrastructure to Develop Complex Distributed Systems*, In Proceedings of the 20th International Conference on Software Engineering, Kyoto, Japan, April 1998.

[33] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh, *WebWork: METEOR$_2$'s Web-based Workflow Management System*, Journal of Intelligent Information Management Systems, pp. 185-215, 1997.

[34] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah, *ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$*, in Proceedings of the 23rd International Conference on Very Large Databases, Athens, Greece, 1997.

[35] A. Sheth, K. J. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shevchenko. *Supporting State-Wide Immunization Tracking using Multi-Paradigm Workflow Technology*, In Proceedings of the 22nd International Conference on Very Large Data Bases, Bombay, India, September 1996.

[36] Open Hypermedia Systems Working Group (OHSWG) home page. http://www.ohswg.org.

[37] *Proceedings of the 1st Open Hypermedia Systems Workshop (OHS1)*, Edinburgh, Scotland, September 1994.

[38] *Proceedings of the 2nd Open Hypermedia Systems Workshop (OHS1)*, Washington, D.C., March 1996.

[39] *Proceedings of the3rd Open Hypermedia Systems Workshop (OHS1)*, Southampton, UK, April 1997.

[40] *Proceedings of the 4th Open Hypermedia Systems Workshop (OHS1)*, Pittsburgh, PA, June 1998.

[41] *Proceedings of the 5th Open Hypermedia Systems Workshop (OHS1)*, Darmstadt, Germany, February 1999.

[42] K. M. Anderson, R. N.Taylor and E. J. Whitehead, *Chimera: Hypertext for heterogeneous software environments*, in Proceedings of the ACM European conference on Hypermedia Technology (ECHT '94), Edinburgh, Scotland, UK, Sept. 18-23, 1994.

[43] K. Osterbye and U. K. Wiil, *The Flag Taxonomy of Open Hypermedia Systems*, in Proceedings of the 7th ACM Conference on Hypertext, pp. 129-139, Washington DC, USA., March 16-20, 1996.

[44] K. M. Anderson, *Integrating Open Hypermedia Systems with the World Wide Web*, in Proceedings of the 8th ACM Conference on Hypertext, pp. 157-166. Southampton, UK. April 6-11, 1997.

[45] N. O. Bouvin, *Unifying Strategies for Web Augmentation*, in Proceedings of the 10th ACM Conference on Hypertext, pp. 91-100. Darmstadt, Germany, February 1999.

[46] U. K. Wiil and J. J. Leggett, *HyperForm: using extensibility to develop dynamic, open and distributed hypertext system*, in Proceedings of the ACM conference on Hypertext (ECHT '92), pp. 251- 261, Milan, Italy, November 30-December 4, 1992.

[47] A., Rizk and L. Sauter, *Multicard: An open hypermedia system*, in Proceedings of the ACM conference on Hypertext (ECHT'92), pp. 4-10, Milan, Italy, November 30-December 4, 1992.

[48] G. E. Kaiser and S. E. Dossick, *Workgroup Middleware for Distributed Projects*, in Proceedings of the IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'98), pp. 63-68, Stanford CA., June 1998.

[49] A. Chankhunthod, P. B. Danzig, C. Neerdales, M. F. Schwartz, and K. J. Worrell, *A Hierarchical Internet Object Cache*, in Proceedings of the USENIX 1996 Annual Technical Conference, January 1996.

[50] R. Tewari, M. Dahlin, H. Vin and J. Kay, *Design Considerations for Distributed Caching on the Internet*, in Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, Austin, Texas, 31 May - 4 June, 1999.

[51] J. R. Lo Verso and M. S. Mazer, *Caubweb: Detaching the Web with Tcl*, in Proceedings of the 5th Annual USENIX Tcl/Tk Workshop, July 1997.

[52] J.J. Kistler and M. Satyanarayanan, *Disconnected Operation in the Coda File System*, ACM Transactions on Computer Systems 10(1): 3-25, Feb. 1992.

[53] P. Skopp and G. E: Kaiser, *Disconnected Operation in a Multi-user Software Development Environment*, in Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems, 1993.

[54] R. Allen and D. Garlan, *Formalizing Architectural Connections*, in Proceedings of the 16th International Conference on Software Engineering (ICSE'97), pp. 71-80, Sorrento, Italy, May 199.

[55] D.C Luckham and J. Vera, An event-based architecture Definition Language, IEEE Transactions on Software Engineering, 21(9):717-734, September 1995.

[56] R. Medina-Mora et al., *The Action Workflow Approach to Workflow Management Technology*, in Proceedings of Computer-Supported Cooperative Work'92, pp. 281-288, New York 1992.

[57] S. Bandinelli, E. Di Nitto and A. Fuggetta, *Supporting Cooperation in the SPADE-1 Environment*, IEEE Transaction on Software Engineering, 22(12):841-865, December 1996.

[58] C. Frenstrom, ProcessWEAVER, *Adding Process Support to UNIX*, in Proceedings of the 2nd international Conference on Software Process, pp. 12-26, Los Alamitos, CA., February 1993.

[59] N.S. Barghouti, Supporting Cooperation in the Marvel Process-Centered SDE, in Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments, pp. 21-31, Tyson's Corner VA., December 1992.

[60] D. Heimbigner, *The ProcessWall: A Process Server Approach to Process Programming*, in Proceedings of the 5th ACM/SIGSOFT Conference on Software Development Environments, Washington, D.C., 9-11 December 1992.

[61] E. Di Nitto and D. S. Rosenblum, *Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures*, in Proceedings of the 21st International Conference on Software Engineering (ICSE 99), Los Angeles, CA, May 1999.

[62] Object Management Group, *CORBA Services: Common Object Services Specification*, Technical report, OMG, December 1998.

[63] Object Management Group, *The Common Object Request Broker: Architecture and Specifications*, Technical report, OMG, Revision 2.0, July 1996.

[64] Open Software Foundation. *Introduction to OSF DCE Release 1.1*. Prentice-Hall, 1995.

[65] Sun Microsystems, *Java Remote Method Invocation Specification*. Technical Report, Sun Microsystems, Inc. February 1997.

[66] M. Horstmann and M. Kirtland, *DCOM Architecture*, Technical Report, Microsoft Corporation, July 23, 1997.

[67] M. Kirtland, *Object-Oriented Software Development Made Simple with COM+ Runtime Services*, Microsoft Systems Journal, November 1997.

[68] M. Kirtland, *The COM+ Programming Model Makes it Easy to Write Components in Any Language*, Microsoft Systems Journal, December 1997.
[68] P. A. Bernstein, *Middleware: A model for Distributed System Service*, Communication of the ACM, 39(2): 86-98, February 1996.
[69] G. Cugola, P.Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, *Customizing the behavior of middleware: the PIE approach*, in Proceedings of the Workshop on Reflective Middleware (RM 2000), http://www.comp.lancs.ac.uk/computing/rm2000/paper-list.htm, IBM Palisades Executive Conference Center, New York, USA, April 7-8, 2000
[70] B. Rhodes and T. Starner, *The Remembrance Agent: A continuously running automated information retrieval system*, in Proceedings of The 1st International Conference on The Practical Application of Intelligent Agents and Multi Agent Technology (PAAM '96), pp. 487-495, London, UK, April 1996.
[71] N. S. Glance, D. Arregui and M. Dardenne, *Making Recommender Systems Work for Organizations*, in Proceedings of The 4th International Conference on The Practical Application of Intelligent Agents and Multi Agent Technology (PAAM '99), London, UK, April 19-21,1999.
[72] M. Balabanovic, *An Adaptive Web Page Recommendation Service*, in Proceedings of the 1st international conference on Autonomous agents, pp 378 – 385, Marina del Rey, CA., February 5 - 8, 1997.
[73] P. B. Kantor, E. Boros, B. Melamed, V. Meñkov, *The Information Quest: A Dynamic Model of User's Information Needs*, in Proceedings of the ASIS Annual Conference (ASIS'99), Washington, DC, October 31 - November 4, 1999.
[74] H. A. Kautz, B. Selman and M. Shah, *The Hidden Web*, AI Magazine, 18(2):27-36, Summer 1997.
[75] A. Fuggetta, F. P. Picco and G. Vigna, *Understanding Code Mobility*, IEEE Transactions on Software Engineering, 24(5): 342-361, May 1998.
[76] The Bluetooth Special Interest Group, *The Bluetooth Specification*, v. 1.0, http://www.bluetooth.com/developer/specification/specification.asp
[77] J. Widom and S. Ceri. *Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann Publishers Inc., 1996.
[78] K.R. Dittrich, S. Gatziu and A. Geppert. *The Active Database Management System Manifesto: a Rulebase of ADBMS Features.* In Proceedings of the Second International Workshop on Rules in Database Systems, *RIDS'95*, pp. 3-17, Glyfada, Athens, Greece, September 1995.
[79] F. Cattaneo, A. Fuggetta, L. Lavazza and G. Valetto, *Exploiting the Web to Manage Software Development Artifacts: the Labyrinth Project*, in Proceedings of the 2nd Workshop on Software Engineering over the Internet, 21st ICSE, Los Angeles, USA, May 17, 1999.
[80] W. Reese, D. Heimbigner, A. Wolf, *WIT: A Tool for Integrating Web-Accessible Data*, Technical Report CU-CS-887-99, University of Colorado, Boulder, CO. October 1999.
[81] L. Lee, H. S. Nwana, N. R. Jennings, *Co-ordination in Multi-Agent Systems*, in H. S. Nwana and N. Azarmi eds. Software Agents and Soft Computing, LNAI no. 1198, pp. 42-58, 1997.
[82] H. S. Nwana and M. Wooldridge, *Software Agent Technologies*, in H. S. Nwana and N. Azarmi eds. Software Agents and Soft Computing, LNAI no. 1198, pages 59-77, 1997.
[83] H. S. Nwana and D. T. Ndumu, *An Introduction to Agent Technology*, in H. S. Nwana and N. Azarmi eds. Software Agents and Soft Computing, LNAI no. 1198, pages 3-26, 1997.
[84] T. Finin, R. Fritzon, D. McKay, R. McEntire, *KQML as an Agent Communication Language*, in Proceedings of the 3rd International Conference on Information and Knowledge Management, 1994.
[85] J. Rumbaugh, I. Jacobson, G. Booch, *Unified Modeling Language User Guide*, Addison-Wesley, 1998.
[86] J. P. Morgenthal, *Enterprise Messaging with XML*, Component Strategies, May 1999.
[87] P. Heinl and H. Schuster, *Towards a Highly Scaleable Architecture for Workflow Management Systems*, in Proceedings of the 7th International Workshop on Database and Expert Systems Applications (DEXA '96), pp 439-444, , Zurich, Switzerland, September 9-10, 1996.
[88] G. Alonso et al., *Exotica/FMQM: A Persistent Message-based Architecture for Distributed Workflow Management* in Proceedings of the IFIP WG8.1 Working Conference on Information Systems Development for Decentralized Organisations, London, 1995.
[89] I. Ben-Shaul and G. E. Kaiser, *A Paradigm for Decentralized Process Modeling*, Kluwer Academic Publishers, Boston MA, 1995.
[90] D. R. Jeffery and L. G. Votta, *Guest Editor's Introduction to the Special Section on Empirical Software Engineering*, IEEE Transactions on Software Engineering, 25(4): 435-437, July/August 1999.

[91] S. Chulani, B. Boehm and B. Steece, *Bayesian Analysis of Empirical Software Engineering Cost Models*, IEEE Transactions on Software Engineering, 25(4): 573-583, July/August 1999.
[92] K. J. Werkman, *Knowledge-based Model of Negotiation Using Shareable Perspectives*, in Proceedings of the 10th International Workshop on DAI, 1990.
[93] S. Bussmann and J. Muller, *A negotiation Framework for Co-Operating Agents*, in Proceedings of CKBS-SIG, pp. 1-17University of Keele, 1992.
[94] Y. Jin and T. Koyoma, *Multi-agent Planning through Expectation-based Negotiation*, in Proceedings of the 10th International Workshop on DAI, 1990.
[95] H.P. Nii, *Blackboard Systems Part One*, AI Magazine, 7(2): 38-53, 1986.
[96] H.P. Nii, *Blackboard Systems Part Two*, AI Magazine, 7(3): 82-106, 1986.
[97] R. Bisiani and A. Forin. *Multilanguage Parallel Programming of Heterogeneous Machines*, IEEE Transactions on Computers, 37(8):930-945, August 1988.
[98] S. Dami, J. Estublier and M. Amiour, *APEL: a Graphical Yet Executable Formalism for Process Modeling*, in E. Di Nitto and A. Fuggetta eds. Process Technology, January 1998.
[99] J. Estublier, M. Amiour and S. Dami, *Building a Federation of Process Support Systems*, in the Proceedings of the Work, Activity Coordination and Cooperation conference (WACC'98); San Francisco, CA., 22, 26 February 1998.
[100] A. G. Cass, B. Staudt Lerner, E. K. McCall, L. J. Osterweil and A. Wise, *Logically Central, Physically Distributed Control in a Process Runtime Environment*, Technical Report # UM-CS-1999-065, University of Massachusetts, Computer Science Department, Amherst MA., November 11, 1999.
[101] A. G. Cass, B. Staudt Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton Jr., A. Wise, *Little-JIL/Juliette: A Process Definition Language and Interpreter*, to appear in Proceedings of the 22nd International Conference on Software Engineeering (ICSE 2000), Limerick, Ireland, June 4-11, 2000.
[102] IBM, *IBM MQSeries Workflow Concepts and Architecture*, Technical Report, International Business Machine co., July 1998.
[103] *TIB Active Enterprise products*, http://www.tibco.com/products/enterprise.html
[104] DARPA Information Technology Office, *Active Networks*, Solicitation # BAA 99-11, November 6, 1998, http://www.darpa.mil/ITO/Solicitations/CBD_9911.html
[105] G. E. Kaiser, C. Vaill, and S. Dossick, *A Workgroup Model for Smart Pushing and Pulling*, in Proceedings of the 8th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'99), pp. 15-21, June 1999.
[106] G. E. Kaiser, A. Stone, and S. Dossick, *A Mobile Agent Approach to Lightweight Process Workflow*, in Proceedings of the International Process Technology Workshop, Villard de Lans, France, September 1-3, 1999.
[107] D. Lange and M. Oshima, *Programming and Deploying Java™ Mobile Agents with Aglets™*, 1998.
[108] A. K. Elmagarmid (ed.), *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.
[109] M. N. Huhns and M. P. Singh (eds.), *Internet-based Agents: Applications and infrastructure*, special issue IEEE Internet Computing, 1(4), July/August 1997.
[110] J. E. Cook and A. L. Wolf. *Software process validation: Quantitatively measuring the correspondence of a process to a model using event-based data*, ACM Transactions on Software Engineering and Methodology, 8(2): 147-176, Apr 1999.
[111] D. Garlan et al. *Architecture-based Adaptation of Complex Systems*, private communication, February 2000.
[112] D. Heimbigner, *Proscription versus Prescription in Process-Centered Environments*, in Proceedings of the 6th International Software Process Workshop, Hakodate, Japan, 29-31 October 1990.
[113] P. Chen, *The Entity-Relationship Model - Toward a Unified View of Data*, ACM Transactions on Database Systems, 1(1): 9-36,March 1976.
[114] C. Pu, W. Hseush, G. E. Kaiser, K. Wu and P. S. Yu, *Divergence Control for Distributed Database Systems*, in: Distributed and Parallel Databases, Kluwer Academic Publishers, January 1995.

Appendix A; Synoptic tables of the workflows

Synopsis of coordination problems						
Distribution characteristics of the WF						
WF Data	Design time	WF Specification				
		Type system				
	Run time	WF State				
		Artifacts				
		Resources				
WF architecture	Static architecture					
	Dynamic architecture					
WF enactment	Reactivity					
	Proactivity					
Other WF characteristics						

Table 1: template of the synoptic table for WF distribution characteristics.

Terminology

In the following tables we employ some specific terms to qualify the level of distribution along the various dimensions, which are better explained upfront:

- **Unconstrained** distribution: there is no constraint or dependency for distribution dimension in object with respect to any other distribution aspect, or any other consideration relative to the coordination process. Hence, the WF elements relevant to that WF distribution can be freely distributed.
- **Matching** distribution: the characteristics of the WF distribution dimension in object are the same as those of another distribution dimension. For example, if WF agents are constrained to be co-located with the WF definitions of task they are responsible for, their distribution must **match** that of the WF specification dimension.
- **Conforming** distribution: the WF distribution dimension in object is constrained by some external (i.e., not relative to WF distribution considerations) compelling factor in such a way that it must strictly comply with it. As an example, consider the WF dedicated to the run-time control of a target distributed system (the Coordination problem 1 of Section 3.1), in which the WF architecture is constrained to overlay the architecture of the target system.
- **Irrelevant**: the WF distribution in object does not play an important role in the case at hand.

Coordination problem 1: run-time distributed control WF					
Distribution characteristics of the WF					
WF Data	Design time	WF Specification	Unconstrained distribution	freely distributed task declarations	action definitions remotely located wrt declarations
		Type system	Unconstrained distribution		
	Run time	WF State	Task state: matching distribution wrt the executing agents' distribution	Process state: unconstrained distribution	Process state variables and events logs in distributed state servers
		Artifacts	matching distribution wrt the executing agents' distribution	May be irrelevant	
		Resources	matching distribution wrt the executing agents' distribution		
WF architecture	Static architecture		Conforming distribution	Overlaid on top of target system	
	Dynamic architecture		Conforming distribution	Reflects dynamism of target systems	
WF enactment	Reactivity		Completely reactive WF		
	Proactivity		Irrelevant		
Other WF characteristics					
Automation level		Largely automated			
Explicitness		Implicit		Fragmented	Bottom-up contstruction
Repeatability		Highly repeatable			

Table 2: Synoptic table of the characteristics of the control WF.

Coordination problem 2: distributed agents coordination WF				
Distribution characteristics of the WF				
WF Data	Design time	WF Specification	Unconstrained distribution	Co-located task declarations and action definitions
		Type system	Unconstrained distribution	
	Run time	WF State	Task state: matching distribution wrt the executing agents' distribution	Process state is composition of task states
		Artifacts	matching distribution wrt the executing agents' distribution	
		Resources	matching distribution wrt the executing agents' distribution	Impromptu resources present: unconstrained distribution
WF architecture	Static architecture		Unconstrained distribution	
	Dynamic architecture		Unconstrained distribution	
WF enactment	Reactivity		Present (for agent autonomy)	
	Proactivity		Present (by delegation	Present (by automation)
Other WF characteristics				
Automation level		Completely automated		
Explicitness		Goal-oriented		Largely explicit
Repeatability		Highly repeatable (plan)		
Enactment		Very dynamic		Limitedly open-ended

Table 3: Synoptic table of the characteristics of the agent coordination WF.

AI2TV case study application WF					
Distribution characteristics of the WF					
WF Data	Design time	WF Specification	Unconstrained distribution		
		Type system	Unconstrained distribution		
	Run time	WF State	Task state: matching distribution wrt the executing agents' distribution	Task and process state available to the content provision architecture	
		Artifacts	Conforming distribution wrt to the content provision architecture and executing agents' distribution	Non-negligible latency issues	
		Resources	Conforming distribution wrt to the content provision architecture executing agents' distribution	Non-negligible latency issues	Impromptu resources present
WF architecture	Static architecture		Unconstrained distribution		
	Dynamic architecture		Unconstrained distribution	Support of disconnection / reconnection	Support of stakeholders' mobility
WF enactment	Reactivity		present		
	Proactivity		Present (by delegation)	Present (by automation)	
Other WF characteristics					
Automation level		Human-oriented		limited	
Explicitness		Completely explicit			
Enactment		Very open-ended		Likely exceptions	
Temporal concerns		Non-instantaneous availability of certain artifacts/resources		Lower-bounded duration of some tasks	

Table 4: Synoptic table of the characteristics of an application WF for AI2TV.

Appendix B: Synoptic table for the comparison of decentralized WFMSs.

Distribution dimension coverage by the WFMS											
WFMS			Endeavors	OPSS	Serendipity-II	WebWork	Opera	Oz	APEL	Juliette	MQSeries
WF Data	Design time	WF Specification	●	✓	●	●	✓ (Rel. or OO DB)	●	✓	✓	●
		Type system	●	□	?	?	✓ (Rel. or OO DB)	●	✓	?	●
	Run time	WF State	●	□	✓ (duplicated)	●	✓ (Rel. or OO DB)	●	✓	●	✓ (Rel. DB)
		Artifacts	●	●	●	✓	✓ (Rel. or OO DB)	●	✓	●	✓ (Rel. DB)
		Resources	●	●	●	✓	✓	●	✓	●	✓ (Rel. DB)
		Impromptu resources	□	□	□	□	□	□	□	□	□
WF architecture	Static architecture		●	●	●	✓	●	✓	●	●	●
	Dynamic architecture		✓	●	●	□	□	□	✓	●	□
WF enactment	Reactivity		●	●	●	●	●	●	●	✓	●
	Proactivity		●	●	●	●	●	●	●	●	●

Table 5: synoptic table of some decentralized WFMSs.

Legend:

● complete distribution support

✓ partial distribution support

☐ no distribution support

? irrelevant/unknown

Note: it can be observed form the table that Endeavors operates on most of the distribution dimensions; therefore, it can be considered a promising candidate to become the experimental WF platform upon which to investigate the reference problems and the WF-based coordination of distributed SW systems at large, in order to identify what adjustments and extensions may be necessary. Another interesting candidate seems to be Juliette.

Appendix C: Figures

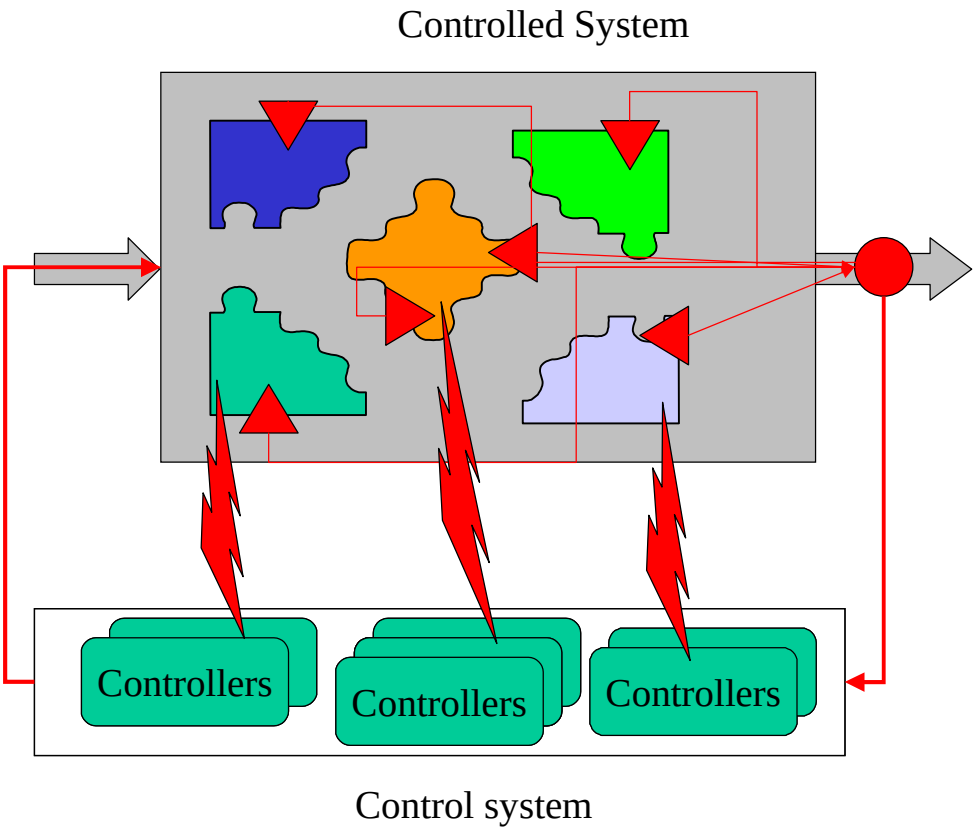

Figure 1: Reactive behavior of a distributed control system.

Note: triangles represent probes; the control is exerted (possibly cooperatively) by the components of the control system (i.e., the Controllers), which elaborate on the data transmitted by probes and operate with directives that influence some single components of the controlled system, or its overall functioning. The red circle represents a facility to compose the elementary “signals” emitted by the probes into semantically richer notifications, of significance for the control system.

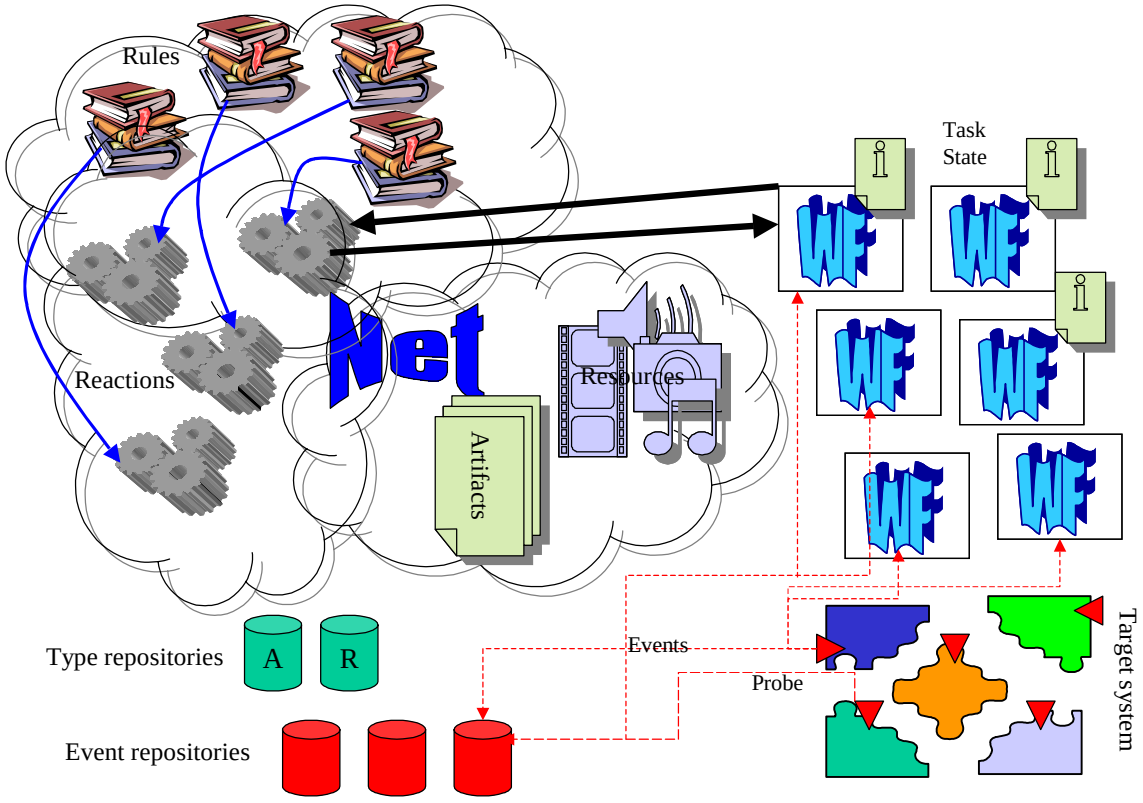

Figure 2: Maximum distribution scenario for the control WF.

For clarification and due to the informality of the representation, notice the following about Figure 2:

1. Each separate element shown in the Figure is intended as being possibly located on a different host with respect to those of all other components, with the exception of probes, which are co-located with the components of the target system or the connectors among them.
2. The "clouds" represent the WWW at large. Elements of the Figure included in the clouds can be dispersed anywhere over the WWW. Elements NOT included in the clouds may indeed still reside and operate over the WWW but are not dispersed anywhere over the WWW; rather, each of them must have a well-known network location (even if the location can change during the enactment of the WF, for instance because of spontaneous networking).
3. Thin solid arrows indicate pointers maintained between rule definitions and their corresponding action, which allow independent distribution of those two parts of the WF specification.
4. Thick solid arrows symbolize a task request by a WF agent and the consequential transfer of the relevant WF specification information. Notice that the request by a WF agent is always a consequence of the reception of an event that matches a rule definition, since the WF is purely reactive (the reactive vs. proactive distribution cannot adequately be represented in the Figure).
5. Dashed arrows indicate event circulation originated by the probes.

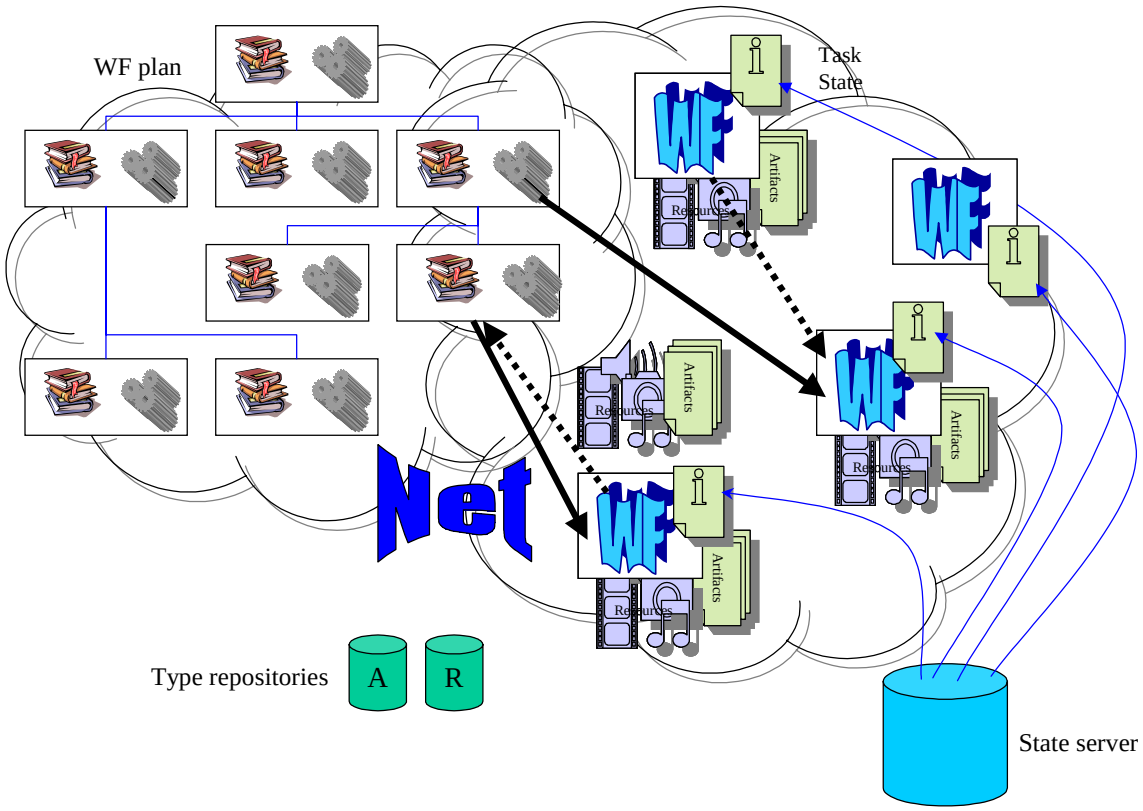

Figure 3: Maximum distribution scenario for the agent coordination WF.

For clarification and due to the informality of the representation, notice the following about Figure 3:

1. Each separate element shown in the Figure is intended as being possibly located on a different host with respect to those of all other components, as for the maximum distribution scenario, with the partial exception of WF agents, which are at times co-located with artifacts and/or resources for reasons of convenience, as discussed above.
2. The "clouds" represent the WWW at large. Elements of the Figure included in the clouds can be dispersed anywhere over the WWW. Elements NOT included in the clouds may indeed still reside and operate over the WWW but are not dispersed anywhere over the WWW; rather, each of them must have a well-known network location (even if the location can change during the enactment of the WF, for instance because of spontaneous networking).
3. Thin solid arrows indicate pointers maintained between fragments of distributed WF information.
4. Thick solid arrows symbolize task requests by WF agents (the dotted arrows) and the consequential transfer of the relevant WF specification information (the solid arrows). Notice that in the Figure, task requests can be initiated by a WF agent on its own, as well as by a WF agent on behalf of another WF agent. This is intended to show that the trade-off and the interplay between guidance and autonomy and reactivity and proactivity in the agent coordination WF.

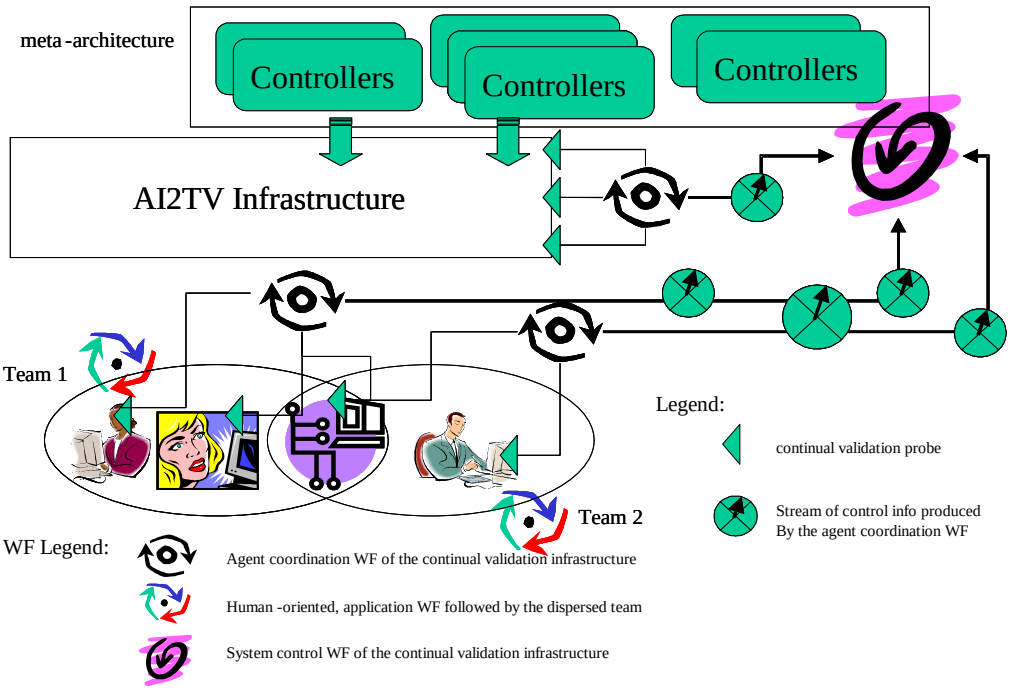

Figure 4: continual validation for AI2TV.

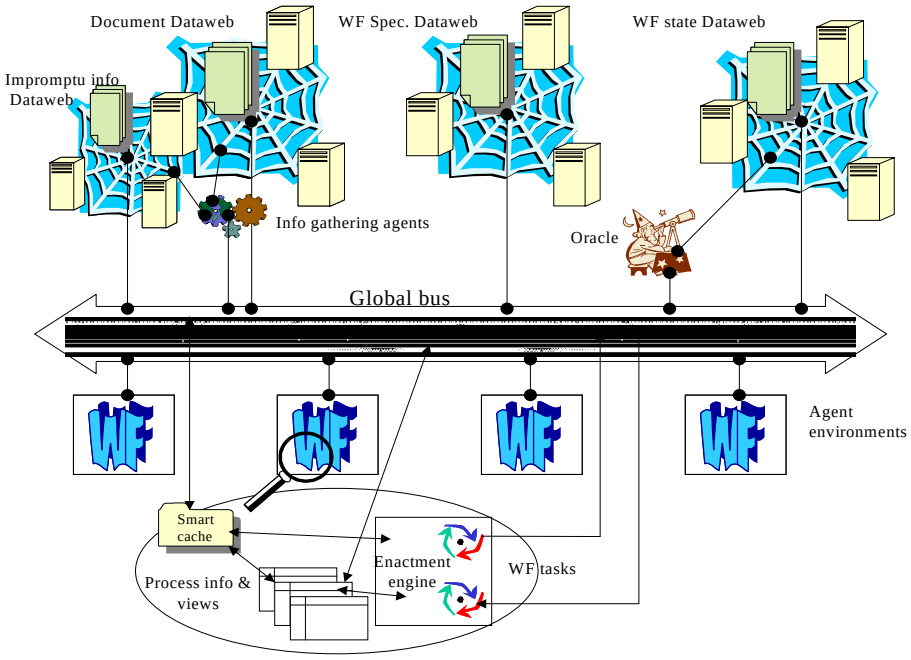

Figure 5: a view of the conceptual WFMS architecture.

Note: in Figure 5 WF agents are positioned separately from datawebs and other parts of the architecture only to augment the clarity of the figure; in fact, as pointed out in Section 2.3, they can be distributed and dynamically re-deployed on any host.

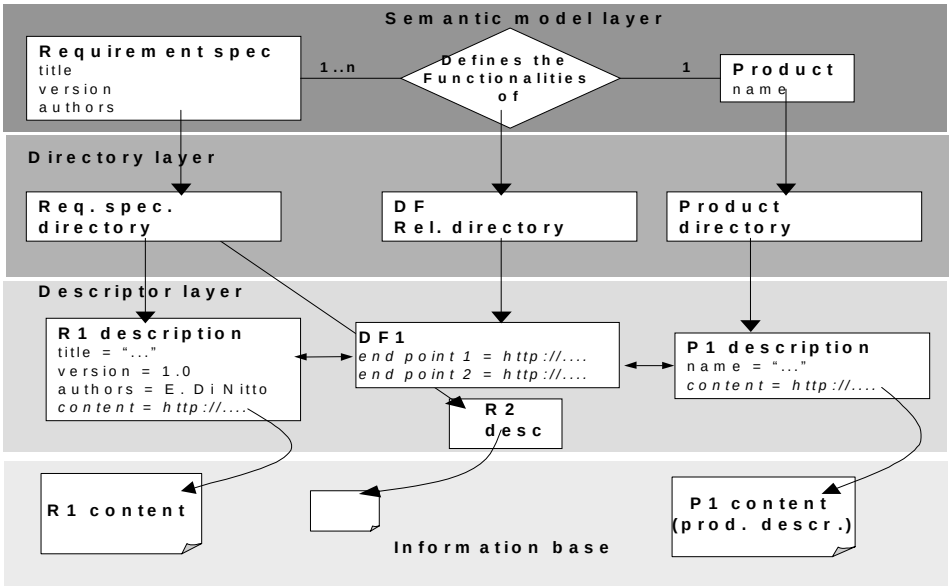

Figure 6: An E/R structured dataweb fragment.

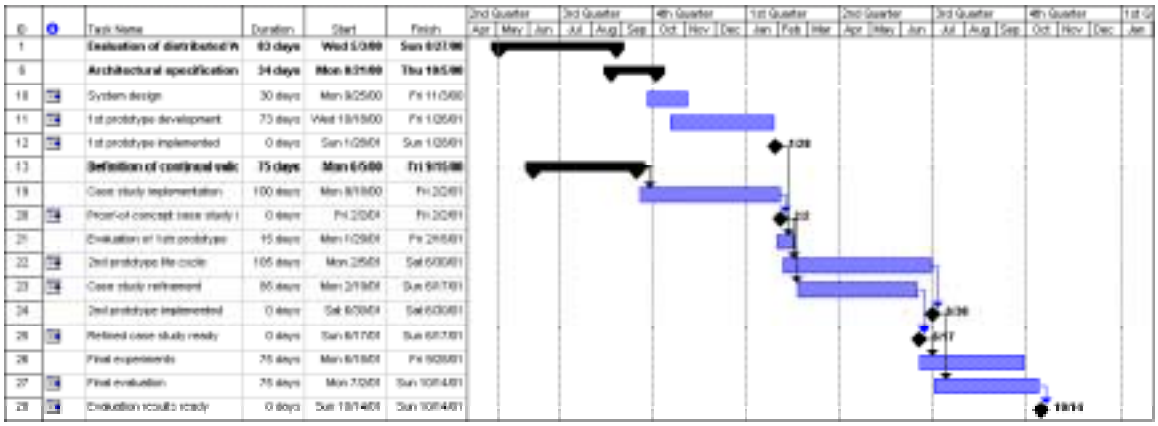

Figure 7: proposed work plan.