

An Internet Indirection Infrastructure - based Workflow Support System

Maksym Petrenko, Monica Brockmeyer

Abstract — This paper proposes peer-to-peer based decentralized network workflow system which is characterized by high mobility and decoupling of the workflow participants as well as the natural simplicity of the underlying model, which is highly important for the typically inexperienced users. Proposed architecture utilizes power of flexible Internet Indirection Infrastructure (*i3*) to organize desired workflow over available IP networks.

Index Terms — WfMS, *i3*, P2P, workflow.

I. INTRODUCTION

Workflow systems are of great interest today. While the client-server based systems are the most developed and supported by now, P2P-based systems promise to outperform them in many aspects like robustness, scalability, reliability etc. Nonetheless, both workflow system types remain too distant from the inexperienced user and require a lot of effort in both system delivery and personal training.

In part, aforementioned problem is caused by the use of the complex and, in many cases, outdated network technologies. The proposed here system takes a view on the architectural and usability opportunities, opened by the second generation of the P2P addressing systems.

Precisely, in this work we demonstrate how to build the simple yet powerful workflow support system on top of the Internet Indirection Infrastructure (*i3*), which is the system, build over the second-generation of P2P addressing systems. Due to this fact, the proposed architecture incorporates the best features of both the P2P addressing systems and *i3* infrastructure delivering simplicity, flexibility and openness.

The rest of the paper is organized in the following way: in section 2 we present motivations of this work; section 3 makes overview of the *i3* P2P system, used in this work; the design of the proposed workflow system is described in section 4; in section 5 we present the additional features of the proposed system; section 6 summarize the features and discusses our system; section 7 concludes and defines the area of future work.

Maksym Petrenko is with the Department of Computer Science, Wayne State University, Detroit, Michigan 48202 (e-mail: max@wayne.edu)

Monica Brockmeyer is with the Department of Computer Science, Wayne State University, Detroit, Michigan 48202 (e-mail: mab@cs.wayne.edu)

II. MOTIVATIONS

Currently, very few P2P-based workflow systems exist (see Related Work section). These systems bring the P2P features to the world of workflow systems; however, they are based on the outdated P2P technologies. Recently, new generation of P2P systems emerged[1] and is based on the concept of DHT (Distributed Hash Table). Basically, DHT is the distribute hash table which allows efficient access to the some peer of interest by the string attribute, associated with this peer.

Couple systems utilize the DHT-based P2P systems. Particularly, *i3* system ([2]) proposes the infrastructure, which utilizes the DHT features to route the data with certain descriptor to the peers, associated with this descriptor. Moreover, in *i3* peers are able to associate themselves with any number of descriptors. Next, each particular descriptor in *i3* serves as the trigger: publishing any data into *i3* with certain descriptor will cause the system to forward immediately these data to the nodes with the same descriptor. Also, *i3* supports some advanced routing schemes such as multicast, anycast and unicast.

On the other hand, workflow systems, the subject of this paper, rout certain process's tasks through the network of workflow participants. Moreover, each task is routed to the participant, which is capable to perform the required task. Typically, the participant's capabilities are expressed in the form of string attributes. In this way, workflow concept of capability naturally maps on the *i3* concept of trigger and workflow concept of process flow maps on the *i3* concept of routing.

This mapping enables us to build the workflow system, which will have the best features of both DHT P2P-based systems namely robustness, scalability, efficiency, stability as well as the features of the *i3* infrastructure namely simplicity, indirect node addressing, nodes' mobility, nodes' decoupling and easy service composition.

III. *I3* OVERVIEW

In this section, we provide a brief overview of *i3* [2]. In a nutshell, *i3* provides indirection, that is, it decouples the act of sending a packet from the act of receiving it. There are two basic operations in *i3*: sources send packets to a logical *identifier* (ID) and receivers express interest in packets by inserting a *trigger* into the network (Figure 1). Triggers can be

thought of as routing entries that point to receivers or to other triggers. Packets are of the form $(id, data)$ and triggers are of the form $(id, addr)$, where $addr$ is either an ID or an IP address. Given a packet $(id, data)$, $i3$ finds the trigger $(id, addr)$, and forwards $data$ to $addr$. Triggers are maintained using soft-state. Receivers refresh their triggers as long as they desire to receive packets sent to those triggers.

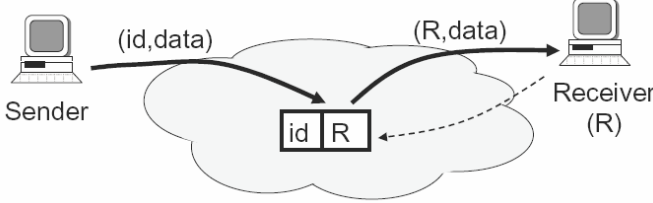


Figure 1. Sending a packet with ID id to receiver R in $i3$.

Identifiers in $i3$ are 256 bits long. IDs in packets are matched with those in triggers using longest prefix matching, where the prefixes are at least 192 bits. $i3$ is implemented as an overlay network of nodes that store triggers and forward packets. Identifiers are mapped to $i3$ nodes using a distributed lookup service such as Chord [3]. A trigger is stored at the node that is responsible for its identifier in accordance to the lookup service. Similarly, packets are routed to the appropriate node using the lookup service.

$i3$ provides the following communication primitives:

- **Mobility.** A mobile host that changes its address from R to R' can preserve the end-to-end connectivity by updating its trigger from (id, R) to (id, R') .
- **Multicast.** Creating a multicast group is equivalent to having all members of the group register triggers with the same ID.
- **Anycast.** All hosts in an anycast group maintain triggers that have identical 192-bit prefixes, but different 64-bit suffixes. Packets are delivered to the group member that has the trigger with the longest matching identifier.

In addition, $i3$ allows either the sender or the receiver to forward packets through intermediate points in the network. One way of performing this in $i3$ is by replacing the packet ID with a stack of IDs. Forwarding such a packet is similar to source routing in IP. Similarly, a receiver can control packet forwarding by replacing the second field of its trigger with a stack that describes the forwarding path. Controlling packet forwarding at the ID level represents the key feature that enables service composition in $i3$.

In general, consider a packet $((id_1, id_2, \dots, id_k), data)$ and a trigger $(id, (id_1', id_2', \dots, id_m'))$. The packet matches the trigger if the first ID in the packet's stack id_1 matches the trigger's ID, id . If they match, the first ID in the stack is replaced by the trigger's stack, and the resulting packet, $((id_1', \dots, id_m', id_2, \dots, id_k), data)$, is forwarded to id_1' .

A. Service composition

Service composition is one of the important $i3$'s features

([4]) and allows building the advanced routes within the $i3$ topology. Figure 2 illustrates the ability of both the sender and the receiver to forward packets through intermediate nodes (middleboxes) that implement third-party services. The sender sends an HTML web page to a PDA identified by ID id . The PDA runs WML, a lightweight version of HTML designed to run on wireless devices with limited capabilities.

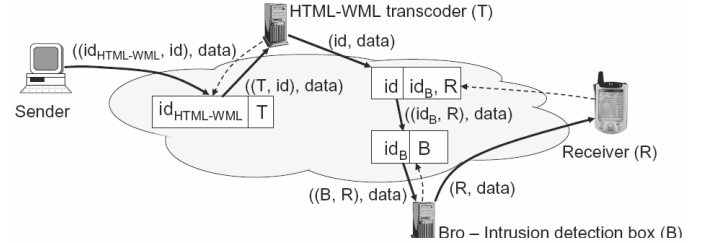


Figure 2. Service composition in $i3$.

The sender uses a stack of IDs $(id_{HTML-WML}, id)$ to forward the packets, first to the middlebox T (which performs HTML-WML transcoding), and then to the receiver. When the packet matches trigger $(id_{HTML-WML}, T)$, the first ID is removed.

The packet's stack $(id_{HTML-WML})$ is replaced with the second field of the trigger (T) , and the resulting packet is forwarded to T . Upon receiving this packet, T performs transcoding, removes the first ID from the stack, and forwards the resulting packet $(id, data)$.

In turn, receiver R uses $i3$ to forward all packets it receives at ID id to an intrusion detection box.

To achieve this, the receiver inserts a trigger $(id, (id_B, R))$, where id_B identifies the intrusion detection middlebox. Upon packet $(id, data)$ matching the trigger, the packet's ID is replaced with the trigger's second field. The resulting packet, $((id_B, R), data)$ is forwarded, via trigger (id_B, B) , to node B and then finally to receiver R .

B. Performance Optimizations

To achieve the best performance, $i3$ uses couple specially designed techniques:

- **Caching.** To improve performance, $i3$ performs aggressive caching. In Figure 1, when packet $(id, data)$ first reaches the $i3$ node storing trigger (id, R) (call that node M) it caches M , and send all subsequent packets directly via IP to M .
- **Nearby triggers.** While caching makes sure that most packets are routed through only one intermediate $i3$ node, that node can be far away from both the sender and the receiver. To alleviate this problem, receivers can sample the ID space, and choose a trigger with an ID, which maps on a nearby $i3$ node. Such triggers are called private triggers. In addition to private triggers, a node (server) that wants to be reachable maintains a public trigger, i.e., a trigger with a well known ID. Once a client contacts a server through its public trigger, they can exchange a pair of IDs which they

use for the remainder of the communication

- **Shortcuts.** Using the above optimizations, every packet is still relayed through at least one *i3* node. While this indirection is necessary for useful functionalities such as multicast and mobility, it is, in general, less efficient than direct IP communication. To address this problem, the communication path between the sender and receiver is allowed to circumvent *i3*. In particular, instead of caching the *i3* node storing the receiver's trigger, a client is allowed to cache the receiver's address itself. The subsequent packets are sent then from source to destination via IP. Note that in this case the *i3* is used as a lookup infrastructure (that resolves an ID to an IP address) instead of a forwarding infrastructure. Shortcuts are used only if both the sender and the receiver allow its use.

IV. SYSTEM DESIGN

As it was stated in the Motivations chapter, our system was designed to utilize the *i3* capabilities to the highest degree without any major changes in the *i3* architecture. Since most of the workflow features naturally maps on those of *i3*, our system's design is very simple and straightforward. In this section we first describe the basic model of the proposed system and then discuss the more advanced workflow structures like sequencing, branching and convergence.

A. Basic Model

The basic model is inferred from the comparative analysis given in the Motivations section. In a nutshell, our workflow system works as following:

- All the peers in organization set up the *i3* triggers for the task they are capable to perform (capability, required for the task, serves as *i3* trigger identifier).
- Workflow administrators can also set-up some additional *i3* service composition triggers for more advanced workflow patterns (see next subchapter for examples).
- If someone needs certain process to be executed, she publishes into the *i3* data with the identifier of the process's initial task.
- Next, published task is routed to the node(s) that is (are) capable to perform this task.
- Those nodes perform the task and, in turn, publish the results of their work back into the *i3* with ID of the next needed task.
- This continues till the process is finished.

Since the trigger ID is the pure string identifier, there is no strict limitations on its use. For the workflow purposes, it is useful to insert the triggers with either node capabilities info or the type of the data this node is interested in.

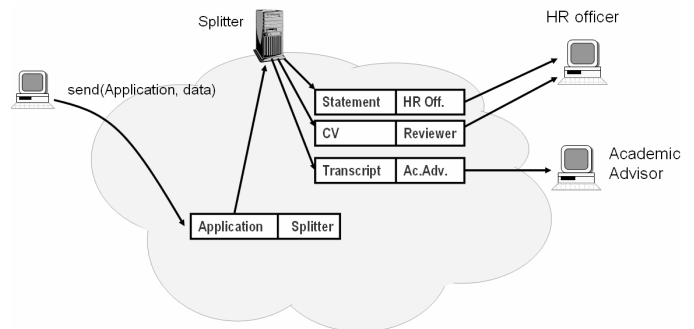


Figure 3. Part of the university's enrollment workflow.

Figure 3 gives the idea of the proposed system's work on example of hypothetical university enrollment process workflow. Thus, student submits his application to the workflow. Automated server 'Splitter', which declared interest in the student applications, receives this application, splits it into the statement, CV and transcript parts and publishes them back into the *i3*. In turn, officer of human resources department receives statement and transcript for review, while academic advisor receives transcript for verification. This receive / perform / publish process continues until the student's application is completely analyzed and the final decision is made.

Please note, that in proposed scheme workflow participants are completely unaware of each other as well as of workflow structure itself. Moreover, once the system of triggers was set, it can be freely reused in any time for any purposes. Basically, it means that if one student used some workflow for the application review, the next student can use the same already existed workflow. Further, any component of the created workflow can be reused in the future. For example, if somebody needs transcript to be verified, she can publish her transcript directly to the *i3* and it will be routed directly to the academic advisor even though advisor's trigger was created to support enrolment process.

As one can see, the proposed model doesn't require any particular process instantiation procedure at all. Also, process is not defined explicitly anywhere in the system; contrary, it exists implicitly in the *i3*'s system of triggers and each part of the process remains in the system for the future reuse until being retracted explicitly by the appropriate *i3* command. Finally, process execution is dataflow-based rather than control-flow based: each task is executed after all the data, needed for this task, were published into the *i3*.

B. Workflow Logic Structures

Typical workflow requires six kinds of logical structures for process execution namely *straight in*, *straight out*, *and in*, *and out*, *or in*, *or out*, which represent various routing structures like sequence, branching and convergence.

In our system, *out*-type structures are supported at level of the *i3* infrastructure, while *in*-type structures are carried by the participating peers themselves.

Thus, if node needs certain task to be processed by the

group of different nodes (*and-out* branching), the node publishes the task to the *i3* as multicast packet. In this case, task will be forwarded to all the nodes, registered for the task's trigger. In more interesting case, there can be more than one node capable to perform the required task (e.g. telephone operators in customer support center). In this case, one should use *i3* service composition capabilities: rather than registering each capable node directly for the trigger of interest, department manager should setup the departmental trigger and then the department members can register for this new trigger (Figure 4). In this way, branched task will be routed to the departmental trigger and then will be assigned for execution to the one of capable nodes.

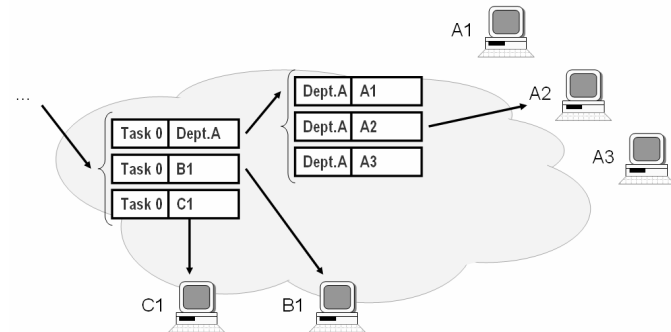


Figure 4. Complex branching.

Similarly, for the *or-out* or *straight-out* node should publish the task to the network as *i3* anycast or unicast packet.

To deal with convergence and sequence workflow structures, each node is able to cache the incoming data and then proceed according to the workflow structure semantic. Thus, in case of *straight-in* node begins task execution immediately after the required packet arrived. In case of *or-in*, node performs the task when any of the required packets arrived. Finally, for the *and-in* node waits for all the required packets of the task and then proceeds.

V. ADDITIONAL DESIGN AND PERFORMANCE ISSUES

Additionally to the very basic workflow functionality, our system proposes some additional features like load balancing, management, process back-up etc. While some features like process management are achieved by connecting to the system some advanced nodes, for some other important properties we utilize the *i3*'s concept of significant and less significant trigger identifier bits. According to this concept, first m bits of the l -bits trigger identifier carry the semantics of the trigger and are used directly to route the packet to the server, which keeps the list of peers, registered for this identifier. The rest $l-m$ bits are utilized to make the local decisions on that server's side in case of anycast routing. In our approach, we divide the $l-m$ by two and then use first half of insignificant bits for the task context routing and second half for load balancing (the second half also serves as unique process identifier). More detailed description and examples are given in subchapters A and B of current chapter.

A. Process Management and Backing-Up

To avoid the loss of process execution results, we connect to the system back-up nodes. Back-Up nodes are regular nodes, connected to the system through the multicast triggers in the critical points of the workflow. Thus, any information, which comes to certain trigger, will be duplicated at the back-up node. Back-up node keeps the captured information during the lifetime of the process. To recover the back-upped information in case of any failure, back-up node is able to republish the saved data back into the workflow infrastructure under the same process identifier. In our system, this will happen if the packet with desired process identifier and certain reserved word (like "republish") as the packet body arrive to the back-up node's trigger and thus is captured and analyzed by this back-up node.

To capture the failures in the workflow, we connect to the network another advanced nodes called managers. Managers behave in the same manner as back-up nodes – they capture and analyze all the data that goes through the critical triggers. Moreover, we suppose each manager is connected to more than one trigger. In this case, manager is connected to the number of critical triggers – checkpoints. In this way, manager is able to verify that certain process passed appropriate checkpoints in the appropriate timeframe. In case of failure, process data will not arrive to the appropriate checkpoint in appropriate timeframe. Thus, manager will capture this problem and will try to resolve it. In the simplest case, manager will ask the back-up node, connected to the last known passed checkpoint, to republish the process data back into the workflow network. Therefore, process data will be routed to the other nodes that remained in the system. In case if there are no required nodes remained in the system, *i3* will notify sender, which in our case is back-Up node, about the problem. Then, back-up node can tell about this problem to the manager by publication of the problem description back to back-up node's trigger. Since manager listens this trigger also, he will be able to capture this information and then try to resolve the problem in any appropriate off-line way. Furthermore, manager can provide the gathered process execution statistic if someone publishes to the workflow network packet with the desired process identifier and certain reserved word (like "statistic") as the packet body.

Figure 5 pictures the example of some workflow's part. In this figure, manager listens the *task 0* and *task 3* triggers, while back-up node captures all the data that comes to the *task 0* trigger. Thus, if *A1*, *A2* or *A3* node fails, the data will not arrive at *task 3* trigger at desired time and manager will ask back-up node to republish the process data back to the *task 0* trigger.

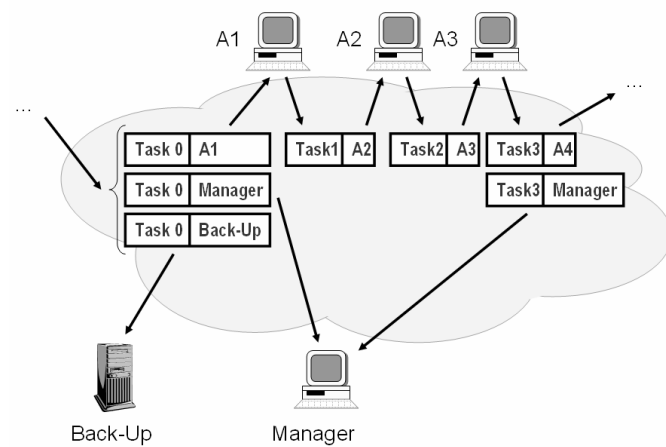


Figure 5. Process management and backing-up.

Please note, that both back-up node and manager node are the regular network nodes and the workflow will be able to function without any single node of these kinds; in this sense, back-up and manager nodes are not the super peers.

B. Task Context

Typically, in organizations each employee is capable of performing required tasks only in the context of certain process. For example, in human resources (HR) department of academic organizations there are some people that are able to process only the applications of the prospective students; also, there some other officers, that are responsible for the applications of the prospective employees. Even though both tasks deal with almost the similar packets of documents, they still require slightly different processing, e.g. for student applicants GPA might mean more then working experience, while exactly opposite for the prospective employees.

In our system, we use first $(l-m)/2$ insignificant bits of the task identifier to denote the context of the task. When some process is published into the $i3$, these bits are assigned accordingly to the process semantics. Then, the process tasks are forwarded to the nodes which are able to perform the requested tasks in the desired context. Figure 6 illustrates the example of application processing, discussed in the previous paragraph. On the picture, triggers' identifiers consist of the string "CV", which denotes the capability needed to perform the task, and of either "student" or "employee" string, which denotes the task context (please note that load balancing bits are not shown on this picture). Thus, if somebody sends to the workflow student's resume for processing, this resume will be forwarded to the S1, which is able to process this resume.

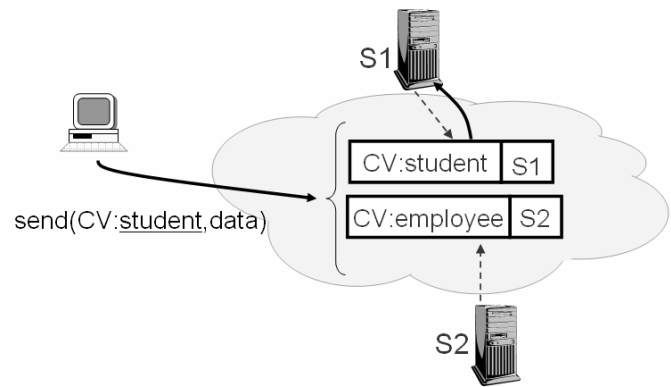


Figure 6. Task context.

However, thanks to the $i3$ closest trigger identifier matching procedure, if there is no exact node for the particular context, the task will be assigned to the node, which can possibly be capable of processing this task. This allows our workflow system to involve in the process auxiliary employees, who will work on behalf of their absent colleges (e.g. sick, fired, etc.). In our example, if S1 is missing, student's CV will be forwarded to the S2, who probably will be able to take care of it.

C. Load Balancing

In current work, we utilize the $i3$ model of load balancing. According to this model, each peer, capable to perform certain task, publishes to the $i3$ trigger, in which last $(l-m)/2$ bits are chosen randomly. Each process, published to the workflow, is also marked with last $(l-m)/2$ random bits. Every time, when there is more then one node capable of performing next process's task, $i3$ server assigns the task to the node, whose random bits sequence is the closest to that one of the task. In this way system load balancing is achieved through random assignment of the tasks to the nodes.

There are also two possible ways to use this idea in the process execution. First, process's initially assigned random bits can be the same for all the tasks of the process. In this way, any two processes, published into the $i3$ with the same random sequence, will be executed by the same nodes' set (except the case, when some nodes of this set quitted the system). This property can be used for the debugging and process monitoring purposes.

Second, each process's task can be marked with the different random sequence. In this case, better randomization of the load can be achieved. However, it will become impossible to repeat the rout of any published process with the known ID.

Figure 7 illustrates the load balancing on the example of two nodes (A and B), both publishing into the $i3$ their CVs for processing (we don't show process's context bits on this picture). Since for both nodes the last 4 bits were randomly chosen, their CVs were forwarded to and processed by the different servers (S1 and S4 respectively). Thus, the load was randomly distributed among the available nodes.

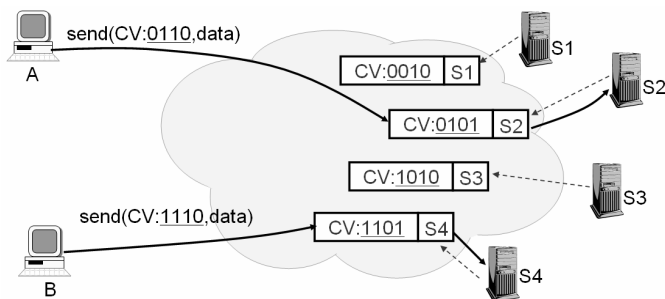


Figure 7. Load balancing.

Nonetheless, proposed model doesn't encounter the load balancing for the nodes, registered for more than one trigger as it was done in SwinDeW [5] system for example. In a nutshell, the problem is that if some node N is registered among some other nodes for some trigger A and in the same time this node is the only node registered for some trigger B , assignment of the task A for the node N will live in the system no nodes capable of performing B at all. In contrast, assignment of the A task to nodes other than N will preserve the N for the B tasks if any. To achieve this kind of balance, we will need to modify $i3$ infrastructure, so the network can detect all the triggers particular node is registered for. In this work we don't address this feature trying to utilize as unmodified $i3$ as possible; therefore, we leave these issues for the future work.

VI. DISCUSSION

In this section we summarize the features of the proposed system for the reader's better understanding of the outcome of our work.

Clearly, the proposed system incorporates the best features of the P2P systems. These features are extensively discussed in the literature ([1]) and thus are not presented here. Also, system incorporates such important features of underlying $i3$ system like nodes' mobility, easy service composition and advanced routing capabilities.

What is even more important, the designed architecture has the number of workflow-specific features. First of all, the system is very simple and open for the end-user. As it was shown in the system design chapter, in the proposed architecture user is totally unaware of the underlying workflow structure. Each user knows just the kind of information he needs for his tasks and kind of information he produces after he performed his tasks. In other words, user knows nothing about any other participants, workflow events, control etc and thus is able to start using the system from scratch without any extensive knowledge gathering.

Next, the workflow is directed by the dataflow rather than the control flow. In the proposed approach, system of the $i3$ triggers is actually the structure that specifies the workflow patterns. As it was discussed earlier, the proposed model doesn't require any particular process instantiation procedure at all and processes are not defined explicitly anywhere in the system; instead, the processes exist implicitly in the $i3$'s system of triggers and each part of the process remains in the

system for the future reuse until being retracted explicitly by the appropriate $i3$ command.

Furthermore, each set of triggers can be supported by the separate physical server. Due to the $i3$ characteristics, failure of one server will not affect the routing processes among other servers. For the workflow, it means that the remaining workflow parts can be freely used despite the other parts' failure.

The proposed system of back-up nodes can also be used to make the workflow system even more fault-tolerant allowing the recovery at the back-up checkpoints.

Next, the designed system has better management properties than other existing P2P workflow systems. Manager nodes, introduced in section 5, allow process execution control, statistic gathering and fault resolution capabilities. Moreover, the proposed management architecture doesn't require workflow participants to report on their progress, thus again making the system more user-friendly.

Finally, our system has the unique feature of fuzzy task assignment (Section 5.B) to the nodes, which can possibly be capable to perform the task even though these nodes were not explicitly registered in the system for the particular task's context.

VII. RELATED WORK

There is the number of works in the area of distributed and / or decentralized workflow systems; however, to our best knowledge there are few works that discuss the workflow support in the context of P2P systems. Two of them are the most related to the work discussed in this paper.

First work, namely SwinDeW [5], was among the first works in the area of P2P-based workflow support architectures. SwinDeW system utilizes the JXTA framework to the highest degree. Thus, participating nodes in the system are organized into the JXTA groups according to their capabilities and are addressed by their group name. Moreover, groups can overlap; thus, each node can be associated with any number of groups. Also, each group is responsible for the assignment of the node, which will carry out a particular task. In the proposed architecture, each process to be executed first needs to be instantiated which means that the needed workflow path is created within the P2P system every time when some process needs to be executed. This is different from the proposed architecture since in our approach each workflow path remains in the system once created therefore saving the path discovery time. Also, in SwinDeW it's hard to control the workflow and handle the exceptions – paper just suggests nodes to periodically report their progress to some super entity thus possibly introducing the single point of failure. Another single point of failure in SwinDeW is introduced by the super peer, which stores the complete workflow process definitions. At last, nodes are forced to keep the part of process's context. In contrast, in our approach nodes are completely relieved of the workflow management tasks such as reporting or task assignment thus making it more desirable for the non-

experienced users.

Second work, done by Fakas[6] et al. utilizes the more recent framework of the WEB-services. In this architecture, nodes should register their capabilities within the WEB-service directory thus providing the services they are capable of. This is somewhat similar to the triggers inserting in our approach; however, Fakas's system relays completely on the underlying WEB-directory of available peers (WWPD). This is similar to the Napster approach, which utilizes the centralized server thus introducing the single point of failure. In contrast, *i3* architecture, which underlay our system, is the next-generation P2P system which overcomes Napster in many parameters and is completely decentralized. Further, in Fakas's system each workflow process is managed by the single peer, which initiated the process. All the participating peers report to this administrator node. Moreover, each participating node carries the whole process definition and thus is in charge of workflow process management. Finally, the [6] work doesn't address the load balancing issues at all leaving them for the future work.

After all, both discussed works have different kinds of super peers and introduce certain single points, which can cause system either to fail or become overloaded. Our approach differs from these two systems through the use of the next-generation P2P overlay technologies as well as the system of distributed and decoupled administrator nodes, which can be assigned or reassigned on-the-fly. Plus, in our system participating nodes are completely decoupled from each other and are totally unaware of the workflow management tasks.

VIII. CONCLUSIONS AND FUTURE WORK

In this work we discussed the architecture and features of the new workflow management system that is based on the *i3* P2P technology. Among the main characteristics achieved in the proposed system we can name extreme simplicity, flexibility, robustness, openness and better management capabilities. In combination with features, offered by the underlying second-generation P2P technology, proposed

system overcomes the other existing P2P workflow solutions and makes the workflow systems closer to the inexperienced end-user.

Next, we plan to investigate the idea of grouping triggers' servers into the functional groups. In other words, *i3* servers that support the triggers, which together compose certain logical functionality, can be grouped together into some logical or physical structures. This should give us opportunity for advanced group-based load balancing (see Load Balancing subchapter). Also, it can potentially improve robustness of our workflow system, since the fault of some functional group will not affect the work of the other functional group. These issues are important for the system yet disputable and therefore should be carefully analyzed in the future.

REFERENCES

- [1] M. H. Karl Aberer, "An Overview on Peer-to-Peer Information Systems," presented at Workshop on Distributed Data and Structures, Kjeller, 2002.
- [2] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," presented at ACM SIGCOMM Conference, 2002.
- [3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," presented at ACM SIGCOMM Conference, 2001.
- [4] K. Lakshminarayanan, I. Stoica, and K. Wehrle, "Support for service composition in *i3*," presented at ACM international conference on Multimedia, 2004.
- [5] J. Yan, Y. Yang, and G. K. Raikundalia, "SwinDeW - A Peer-to-peer based Decentralised Workflow Management System," *IEEE Transactions on Systems, Man and Cybernetics, Part A (accepted for publication)*. 2005.
- [6] G. J. Fakasa and B. Karakostas, "A Peer to Peer (P2P) Architecture for Dynamic Workflow Management," *Information and Software Technology Journal*, vol. 6, pp. 423-431, 2004.