A Versatile Generalized Inverted Kinematics Implementation for Collaborative Working Humanoid Robots: The Stack of Tasks

Nicolas Mansard [1,3], Olivier Stasse [3], Paul Evrard [2,3], Abderrahmane Kheddar [2,3]

[1]*CNRS-LAAS, France,* [2]*CNRS-LIRMM, France*
[3]*CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT*
$nmansard@laas.fr, \{olivier.stasse, paul.evrard, abderrahmane.kheddar\}@aist.go.jp$

***Abstract*— This paper present a framework called the Stack Of Tasks (SoT) implementing a Generalized Inverted Kinematics. This particular implementation provides a run-time graph of computational nodes. It can be modified through a specifically targeted scripting language. It allows hybrid control scheme necessary for complex robot applications such as a HRP-2 humanoid robot in a collaborative working environment. We also show through a case study that this framework allows an efficient integration in nowadays middleware such as CORBA.**

I. Introduction

The Generalized Inverted Kinematics (GIK) introduced by Nakamura et al. [1] to control redundant robots is widely used in humanoid robotics [2][3], as well as its counter part in the force domain the Operational Space approach [4]. Based on the notion of task [5], priority between tasks is introduced by projecting the tasks with lower priority in the kernel of tasks having a higher priority. Initially considering only two tasks the work of Nakumura has been extended by Siciliano [6] to an iterative scheme shortly presented below.

Software projects to control robots already exist, such as ViSP [7] dedicated to provide all the tools necessary to realize visual servoing. Other projects such as Orocos [8] provides a framework and tools to build robots controllers. The Stack of Tasks is mostly dedicated to implement the GIK formalism in an efficient manner. The scripting interface allows a simple and fast bounding to several components based approach. In this paper, this is illustrated through a binding to the OpenHRP control architecture and CORBA servers.

After introducing the GIK formalism in Section II, the Stack Of Tasks framework is detailed in Section III. A case study presented in Section IV illustrates the use of this framework in a complex application were the robot interacts with a human.

II. Stack of Tasks

A. Task definition

Let $\mathbf{q}$ be the vector of the robot articular positions. Let $\mathbf{e_i}$ be a task. Its Jacobian $\mathbf{J_i}$ is defined by:

$$\dot{\mathbf{e}}_\mathbf{i} = \frac{\partial \mathbf{e_i}}{\partial \mathbf{q}} = \mathbf{J_i}\dot{\mathbf{q}} \tag{1}$$

t$\mathbf{J_i}$ is assumed to be of full rank. Assuming that the robot is controlled using $\dot{\mathbf{q}}$, we can compute its value using:

$$\dot{\mathbf{q}}_i = \mathbf{J_i^+}\dot{\mathbf{e}}_\mathbf{i}{}^* \tag{2}$$

where $\dot{\mathbf{e}}_\mathbf{i}{}^*$ is the desired motion in the task space, and where $\mathbf{J_i^+}$ is the pseudo-inverse of $\mathbf{J_i}$. The motion is usually constrained to follow a differential equation:

$$\dot{\mathbf{e}}_\mathbf{i}{}^* = -\lambda \mathbf{e_i} \tag{3}$$

Thus the control law is:

$$\dot{\mathbf{q}}_i = -\lambda \mathbf{J_i^+}\mathbf{e_i} \tag{4}$$

Finally a task $\mathbf{e_i}$ is defined as a difference between a desired feature $\mathbf{s_i^*}$ and its current value $\mathbf{s_i}$:

$$\mathbf{e_i} = \mathbf{s_i} - \mathbf{s_i^*} \tag{5}$$

The current value of the feature and the velocity $\mathbf{v}$ of a point on the robot are usually related by the equation:

$$\dot{\mathbf{s}}_\mathbf{i} = \mathbf{L}_{\mathbf{s_i}}\mathbf{v} \tag{6}$$

where $\mathbf{L}_{\mathbf{s_i}}$ is called the interaction matrix in the field of visual servoing. We finally get the task Jacobian $\mathbf{J_i}$ according to the robot articular Jacobian $\mathbf{J_q}$, and the interaction matrix:

$$\mathbf{J_i} = \mathbf{L}_{\mathbf{s_i}}\mathbf{M}\mathbf{J_q} \tag{7}$$

where $\mathbf{M}$ is matrix expressing the velocity $\mathbf{v}$ from $\mathbf{J_q}$.

One can remark that according to (4) and (5) a task is mostly defined by the feature it is handling $(\mathbf{s_i}, \mathbf{s_i^*})$ and its gain λ. The Jacobian (7) is then simply computed from the interaction matrix provided by the feature and the articular Jacobian of the robot.

B. Handling set of tasks

Let $(\mathbf{e_1}, \mathbf{J_1})$ ... $(\mathbf{e_n}, \mathbf{J_n})$ be n tasks. The control law computed from these n tasks should ensure the priority, that is the task $\mathbf{e_i}$ should not disturb the task $\mathbf{e_j}$ if $i > j$. A recursive computation of the joint velocity is proposed in [6]:

$$\begin{cases} \dot{\mathbf{q}}_\mathbf{0} = 0 \\ \dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_{\mathbf{i-1}} + (\mathbf{J_i}\mathbf{P}^\mathbf{A}_{\mathbf{i-1}})^+(\dot{\mathbf{e}}_\mathbf{i} - \mathbf{J_i}\dot{\mathbf{q}}_{\mathbf{i-1}}), \quad i = 1..n \end{cases} \tag{8}$$

where $\mathbf{P}^\mathbf{A}_\mathbf{i}$ is the projector onto the null-space of the augmented Jacobian $\mathbf{J}^\mathbf{A}_\mathbf{i} = (\mathbf{J_1}, \ldots \mathbf{J_i})$. The robot joint velocity

realizing all the tasks in the stack is $\dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n}$. The projector can be recursively computed by

$$\mathbf{P_i^A} = \mathbf{P_{i-1}^A} - (\mathbf{J_i P_{i-1}^A})^+ \mathbf{J_i P_{i-1}^A} \quad (9)$$

III. Software Framework

A. Entities and graph of entities

At time t one control iteration has to be performed. For each active task the system computes the error related to a task. For this it is necessary to compute the feature $\mathbf{s}(\mathbf{q}(t), t)$ related to the robot state at time t. For some tasks the desired feature value $\mathbf{s}^*$ also depends on t. An efficient system should implement a mecanism which ensure that a value is not computed twice. The solution implemented in the Stack of Tasks is to consider computational unit called *Entities* which provides and consumes signals. A signal providing information is called an *input signal*, and a signal consuming information is called an *output signal*. An output signal is linked with an internal method of the entity which computes the needed information. An output signal can provide its information to any input signals. An input signal is linked with one output signal. The relation between an input and an output signal is specified by a scripting language described in the next paragraph. Signals are time dependant and trigger computation when an entity access a signal input asking for a data which is after the last evaluation.

Each entity is created through a plugin mechanism. First a dynamic library is loaded providing a class of entity, then following a factory design pattern it is possible to create on the fly instances of this entity. Following the formal description given in the previous section, among the available entities two special classes are explicitly handled by the framework: *tasks* and *features*.

In addition to the graph of signals more complex relationships between the entities are provided in this control framework. It is for instance the case with the **StackOfTasks** object which relates tasks using the priority mechanism specified previously. The mediator also allows to provide interfaces for diverses componentization tool. At this current stage, a CORBA server provides the possibility to interact with the StackOfTasks by creating, reading and sending signals as well as sending script commands. Other proxies to component based framework could be used such as GeNom.

B. Scripting

The scripting interface purpose is mostly to handle the underlying framework. The main idea behind this interface is to offer a mean for controlling the framework, without developing a full featured language for which it already exists numerous alternatives. Moreover the script is mostly limited to basic operations. New functionalities are added when loading the plugins. This extend naturally the possibilities of the script within the frame of the SoT. While designing this scripting interface, one of the major goal was to minimize the external interface of the framework and make it resilient to internal algorithmic changes. For instance, it does not make sense to recompile a client using this framework because a new feature has been implemented with some specific new **control** parameters.

1) Factory of entities: More precisely it allows to load classes of entities using dynamic libraries (**loadPlugin** and **unloadPlugin**), create and destroy entities (**new** and **destroy**), run scripts (**run**), and finally triggers computation. The entity producing entities is the **pool**.

2) Entity: An entity provides methods which takes string arguments and convert them internally to appropriate formats. The methods either change some internal states of the entities or send back a stream of strings. For the user to know the methods provided by an entity in interactive mode, the entity creator can provide a help method listing the other methods provided by the entity. For instance typing

```
pool.help
```

returns:

```
Pool:
  - list
  - listFeature
  - listTask
  - writegraph FileName
```

The first method list all the entities created in the current instance of the factory. The second lists only the features, while the third provides the name of the task entities. The last one finally generates dot graph which can be displayed as presented in Fig.1.

3) Signals: Entities can communicate with each other through signals of the same type. The connection of the signals is done through the **plug** command. It is also possible to set and get signals values or references using **set** and **get**. The following paragraphs gives some examples. Signals integrate a temporal dependency which allow to trigger computation only when it is needed. In the graph depicted in Fig.1, the node OpenHRP is an entity and an OpenHRP plugin computed every 5 ms. It has an input node asking a command to an entity of type StackOfTasks.

4) Features: Features objects provide the vectors $\mathbf{s_i}$, $\mathbf{s_i^*}$ and the matrix $\mathbf{L_{s_i} M J_q}$ for a task $\mathbf{e_i}$. The role of the features here is strictly limited to:

- receiving the desired values, the current value of the feature according to the robot state, and the robot articular Jacobian in the proper reference frame i.e. $\mathbf{M J_q}$.
- compute the feature Jacobian $\mathbf{L_{s_i}}$.

Thus a feature used to control the center of mass (CoM) will be expressed as:

```
new FeatureGeneric featureCom
plug dyn.com featureCom.errorIN
plug dyn.Jcom featureCom.jacobianIN
```

where the entity **dyn** provides the current CoM of the robot with the signal **com**, and the CoM Jacobian with the signal **Jcom**.

For instance here is a simple example that creates a desired feature of the robot's CoM and specify a fixed value:

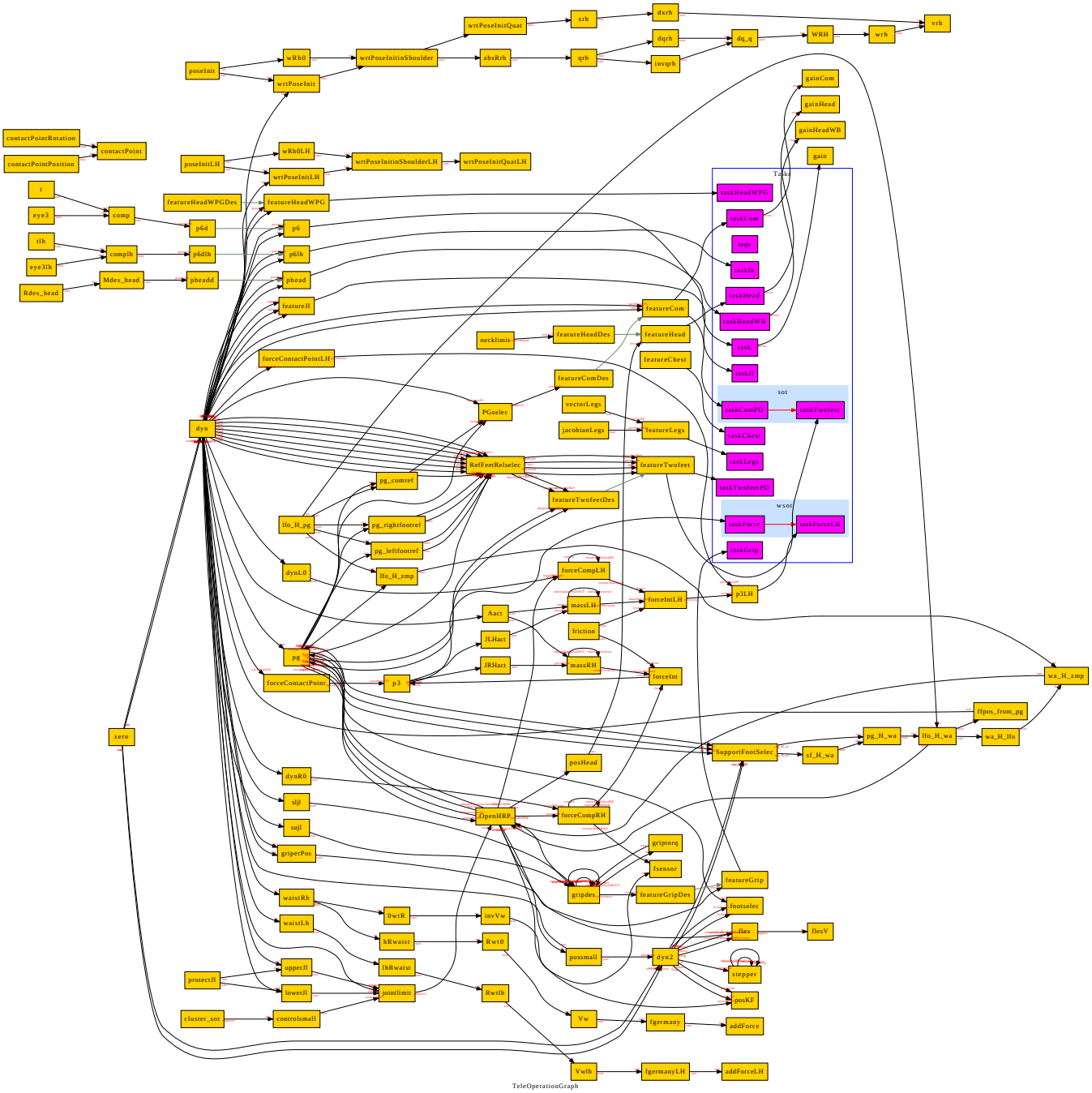

Fig. 1. Graph of entities at a given time of the control.

```
new FeatureGeneric featureComDes
set featureComDes.errorIN [3](0,-0,0.64)
```

To relate this desired reference to the current state of the robot for computing the control:

```
set featureCom.sdes featureComDes
```

To specify a more complex desired CoM, it is possible to plug another feature inside the desired one:

```
plug PGselec.scomref featureComDes.errorIN
```

The current implementation provide several features such as generic ones, 6D position (*i.e.* position and orientation), relative 6D position and visual points.

5) Tasks: A task uses the features provided by the graph to compute a control law:

```
new Task taskCom
taskCom.add featureCom
```

To aggregate several elementary tasks $\mathbf{e_i}$, it is possible to add several features to a task. Indeed to have a task $\mathbf{e_{1,2}}$ which

concatenates $\mathbf{e_1} = \mathbf{s_1} - \mathbf{s_1^*}$ and $\mathbf{e_2} = \mathbf{s_2} - \mathbf{s_2^*}$ such that:

$$\mathbf{e_{1,2}} = \begin{bmatrix} \mathbf{s_1} - \mathbf{s_1^*} \\ \mathbf{s_2} - \mathbf{s_2^*} \end{bmatrix} \tag{10}$$

adding the two features to the same task with the **add** method is enough.

6) Stack of tasks: The entities of type **SOT** compute the control law following eq.(8) considering the tasks added in their stack and their order.

Therefore it is possible to push a task in the stack with:

```
sot.push taskCom
```

It can be removed either by using **pop** if it is at the top of the stack, or by using **rm**. The priority of a task can be changed using **up** and **down**.

7) Entity for robot dynamical model: One entity is in charge of providing the current state of the robot. By reading the sensors values, and applying the Newton-Euler algorithm, articular Jacobian, CoM, ZMP, position, torques and positions can be computed. In order to avoid unnecessary computation, the user specifies the needed operational points and plug the associated information with the desired features. Frame transformation are handled through small entities able to perform basic matrix and vector computations. Here is an example to create two operational points:

```
dyn2.createOpPoint rleg 6
dyn2.createOpPoint lleg 12
```

The number is the rank of the joint in the state vector of the robot. The creation of the operational point creates signals to read position and articular Jacobian to be used by features:

```
new FeaturePoint6dRelative featureTwofeet
plug dyn.Jrleg  featureTwofeet.Jq
plug dyn.Jlleg  featureTwofeet.JqRef
plug dyn.rleg  featureTwofeet.position
plug dyn.lleg  featureTwofeet.positionRef
```

8) Entity for walking: One entity, called **pg** in Fig.1, provides references values for the HRP-2 to walk. This entity can work in two modes: providing leg articular reference values, or providing CoM and feet trajectories. In this work, the former mode is used, and a task to realize the articular references is present inside the graph (**taskLegs**). The humanoid walking pattern generator used to implement this entity is described in [9]. The framework to generate the references trajectories is an earlier version of this scripting system. To interact with the pattern generator a method called **parsemcd** is used. For instance to specify the single support time:

```
pg.parsecmd :singlesupporttime 0.780
```

C. Componentization

In order to externalize the Stack of Tasks, two entities act as proxies to compenentization frameworks. The first one is the **OpenHRP** entity which allow to interact with the OpenHRP architecture introduced by Kanehiro et al. [10] for controlling robots such as the humanoid robot HRP-2. OpenHRP is based upon CORBA and proposed several CORBA servers dedicated to dynamical simulation, collision detection, loading robot models and control architecture. The server dedicated to control is based upon a list of controllers which can be dynamically loaded and created. Its main limitation is the relationship linking the controllers together through a sequential evaluation.

The second one is **coshell** which provides a CORBA server for externals programs to interact with the Stack of Tasks as depicted in Fig.2. The IDL interface provided by this server is:

```
interface SOT_Server_Command
{
  typedef sequence<double> DoubleSeq;
  typedef sequence<char> CharSeq;
  typedef sequence<char> StringStreamer;

  void run( in CharSeq cmd );
  void runAndRead( in CharSeq cmd,
                   out StringStreamer os );
  void readVector( in CharSeq signalName,
                   out DoubleSeq value );

  long createOutputVectorSignal
       ( in CharSeq signalNameCorba );
  long createInputVectorSignal
       ( in CharSeq signalNameCorba );
  void readInputVectorSignal
       ( in long signalRank,
         out DoubleSeq value );
  void writeOutputVectorSignal
       ( in long signalRank,
         in DoubleSeq value );
};
```

The two methods **run** and **runAndRead** allow to execute script commands. The former one do not wait for answer, while the other one returns the answer as a stream of strings. The method **readVector** acts as probe and write the value of the signal specified by signalName with the type vector in **value**. The four last methods make possible to create input and output vector signals for writing and reading informations. Those methods have been used in the experiment presented in paragraph IV-D to synchronize the decision layer and the end of the walking execution. In [11] it was used extensively to realize a wide-range telepresence experiment with a HRP-2 humanoid robot.

IV. Case study

A. Integrating a humanoid robot in a collaborative environment

In [12] we described the integration of the HRP-2 humanoid robot inside a collaborative working environment called BSCW [13]. The robot is able to pull tasks from this environment initially targeted for human users and interpret them to perform a task in the real world. The work in [12]

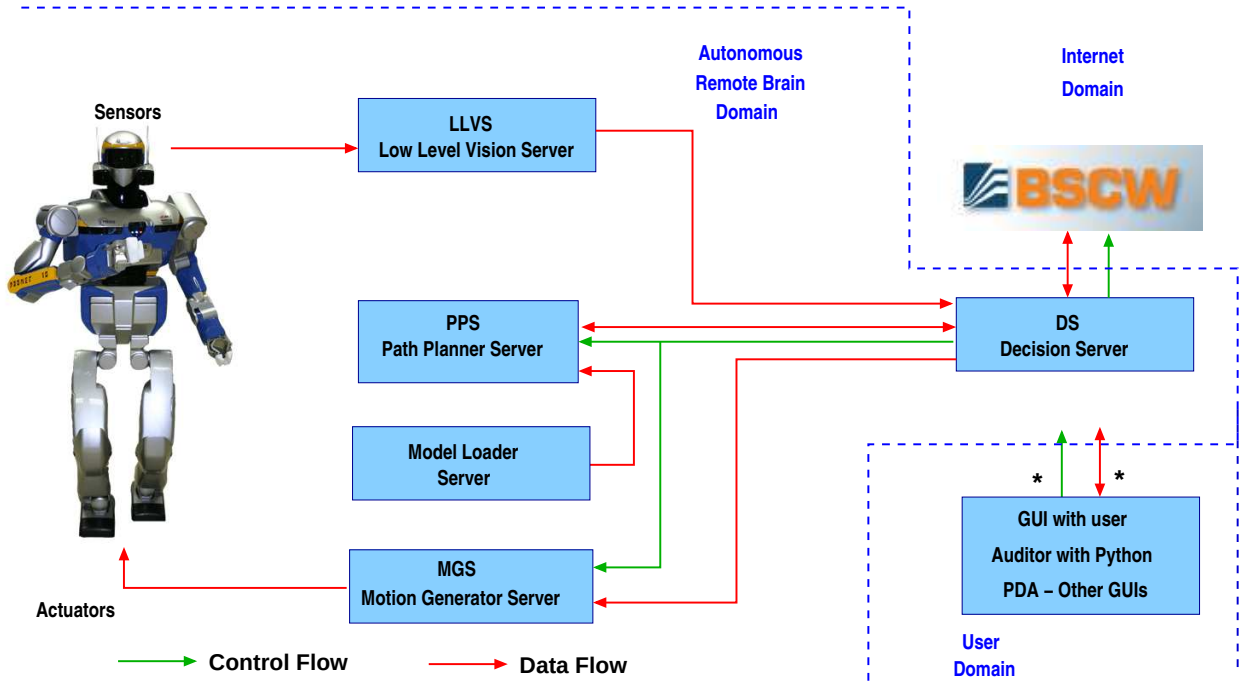

Fig. 2. Functional block implemented as CORBA and OpenRTM servers

presented a specific case where the robot was mostly asked to go to one place, take a picture and fill a report to the requester on BSCW. We present here an extension where the robot is then asked to switch to an interaction mode with the user.

The software architecture used during this experiment is the one depicted in Fig.2. They are 4 CORBA servers providing vision information (**LowLevelVisionServer**), planning (**PathPlannerServer**), decision making (**DecisionServer**), and motion generation(**MotionGeneratorServer**). Each of this server implements a framework specific to the robotic problems solved. For instance, the planner is using KineoWorks which provides all the necessary tools to solve complex planner problems such as the A380 grand itinerary [14], or humanoid robots paths [15]. We also described one framework for walking pattern generation in [9].

B. SoT

The graph of entities used in our StackOfTasks module is depicted in Fig.1. The blue box represents a set of entities which are tasks and that can be handled by an entity of class **StackOfTasks**. The red arrows in the blue box indicates the priorities between the tasks. The green-gray arrows display the relationship between the desired features and the features. This figure represents the state of the graph at a given time, but it can change according to the script command send either by the local interpretor, or through the CORBA server. Another possibility could be a local Finite State Machine as described by Mansard et al. [16] implemented through an entity. The tasks present in the subgraph sot indicates which task are currently taken into account for the control computation. In this specific experiment the task **taskLegs** makes the robot walks. The desired features are articular values for the complete body provided by the pattern generator. It is possible to take into account only the legs, or the full body. Here the full body is used. While walking the **DecisionServer** tests every 1s through the CORBA server if the entity **pg** is still providing reference values. Once this is not the case, the server removes the task **taskLegs** and pushes **taskForce** and **taskForceLH**. Those two tasks are based on the generic controller described in the next paragraph IV-C.

C. Force-Based Control

The generalized inverse kinematics formalism presented in the first section allows the realization of many tasks in the free space. However, for constrained tasks such as tasks involving contact or force-based control tasks, the redundancy formalism does not directly apply. A solution to this problem is the *dynamic inverse* as proposed in [17], [18] (rather than using the *kinematic inverse* (2)). However, this solution requires the robot to be torque controlled, which is not the case for our robot. Hence we propose a solution to obtain a similar behavior based on the kinematic inverse.

The space in which the control is designed is the operational space (6D position) of the contact point denoted by $\mathbf{r}$. On position-controlled robots, a common scheme for force-based control is admittance control, which defines the dynamic behavior of the contact point in response to the contact forces through a differential equation. Generally, the selected dynamics for the contact point is the dynamics of a mass-damper system:

$$\mathbf{M}\ddot{\mathbf{r}} + \mathbf{B}\dot{\mathbf{r}} = \mathbf{f} \tag{11}$$

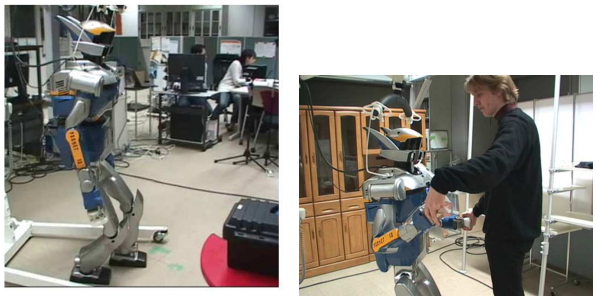

Fig. 3. Left: Robot HRP-2 walking using the SoT Right: Interaction between human and HRP-2

where $\mathbf{M}$ and $\mathbf{B}$ are arbitrary mass and damping matrices of the equivalent virtual system, and $\mathbf{f}$ are forces and torques exerted on the contact point, measured by a force sensor.

In (2), the matrix $\mathbf{J}^{\#}$ can be chosen as *any* pseudo-inverse. We now choose to apply a weighted pseudo-inverse [19], defined when $\mathbf{J}$ is full-row-rank by

$$\mathbf{J}^{\#\mathbf{W}} = \mathbf{W}^{-1}\mathbf{J}^{\mathbf{T}}\left(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^{\mathbf{T}}\right)^{-1} \tag{12}$$

with $\mathbf{W}$ an arbitrary invertible matrix that represents the weights on the joints. Taking the derivative of (2) (with the task $\mathbf{e}$ set to $\mathbf{e} = \mathbf{r}$) and introducing (12) yields:

$$\ddot{\mathbf{q}} = \mathbf{W}^{-1}\mathbf{J}^{\mathbf{T}}\left(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^{\mathbf{T}}\right)^{-1}\mathbf{M}^{-1}\left(\mathbf{f} - \mathbf{B}\dot{\mathbf{r}}\right) \tag{13}$$

Selecting the weight $\mathbf{W}$ as the inertia matrix $\mathbf{A}$ of the robot and the virtual mass $\mathbf{M}$ as $\mathbf{\Lambda} = \left(\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^{\mathbf{T}}\right)^{-1}$, the apparent inertia of the end-effector of the robot, we obtain:

$$\mathbf{A}\ddot{\mathbf{q}} = \mathbf{J}^{\mathbf{T}}\mathbf{f} - \mathbf{B}_{\mathbf{q}}\dot{\mathbf{q}} \tag{14}$$

where $\mathbf{B}_{\mathbf{q}} = \mathbf{J}^{\mathbf{T}}\mathbf{B}\mathbf{J}$ is the friction factor of the whole body structure. This last equation corresponds to a simplified version of the dynamic equation of the whole-body compliant robot with gravity compensation, with forces $\mathbf{f}$ acting on $\mathbf{r}$ and a friction $\mathbf{B}_{\mathbf{q}}$ that may be tuned by selecting $\mathbf{B}$ to stabilize the system.

Consequently, if the control parameters are chosen as described above, the obtained control is equivalent to the real dynamics of the robot. The control represents a generalization of (8), which means that both force-based contact tasks and position-based free space tasks can be realized within this control structure. No specific values have to be chosen or tuned, except for the damping gain $\mathbf{B}$, that has been experimentally verified to be very robust.

D. Experiments

Fig. **??** depicts the robot walking using the task **taskLegs** which assign to the overall control space the articular value to realize. During all the execution as explained earlier, the entity **pg** indicates to the DecisionLayer CORBA server that the walking is being realized. Once this is not the case anymore the DecisionLayer trigger a transition to another state, and run a script changing the tasks and rewiring the graph of entities to perform a compliant based interaction task with a user as depicted in Fig.3. The controller used

Fig. 4. Robot HRP-2 walking and interacting with a human

is the one described in the previous paragraph. The time measurement indicates 300 nanoseconds of overhead induced by the local graph structure. The most time consuming part is the computation of the SVD to evaluate the pseudo-inverse. We have used recently exactly the same architecture to perform a tele-presence experiment [11] with HRP-2 walking as depicted in Fig.4. In this set-up the robot was using the task **taskLegs** using only the 12 DOFs of the legs, and with the tasks **taskForce** and **taskForceLH**. For the latter tasks a signal provided by the remote operator was used in the computation of the final control law. This addition was realized using the CORBA interface described in paragraph III-C. The computation of this control law takes about 3 ms on HRP-2.

V. Conclusion

We have presented a versatile implementation of the Generalized Inverted Kinematics for a humanoid robot HRP-2 called the Stack Of Tasks. This implementation allows dynamic modification of the graph of computation, reference generation, task switching thanks to a scripting language specifically target for this purpose. It has been integrated in a complex architecture to realize collaborative work. Several modalities have been presented: interaction through a classical IT Collaborative Working Environmnent (BSCW), force-control based interaction with a human, force-control based interaction with a human while walking.

Acknowledgment

This work is supported by grants from the ROBOT@CWE EU CEC project, Contract No. 34002 under the 6th Research program `www.robot-at-cwe.eu`.

References

[1] Y. Nakamura and H. Hanafusa, "Optimal redundancy control of robot manipulators," *International Journal of Robotics Research*, vol. 6, no. 1, pp. 32–42, 1986.

[2] M. Gienger, H. Janßen, and C. Goerick, "Task-oriented whole body motion for humanoid robots," in *IEEE/RAS Intl. Conf. on Humanoids Robot.*

[3] N. E. Sian, K. Yokoi, S. Kajita, F. Kanehiro, and K. Tanie, "A switching command-based whole-body operation method for humanoid robots," *IEEE/ASME Transactions on Mechatronics*, vol. 10, no. 5, pp. 546–559, 2005.

[4] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *International Journal of Robotics Research*, vol. 3, no. 1, pp. 43–53, 1987.

[5] C. Samson, M. Le Borgne, and B. Espiau, *Robot Control: the Task Function Approach*. Clarendon Press, Oxford, United Kingdom, 1991.

[6] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *IEEE Int. Conf. on Advanced Robotics (ICAR'91)*, Pisa, Italy, Juin 1991, pp. 1211–1216.

[7] E. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics and Automation Magazine*, vol. 12, no. 4, pp. 40–52, December 2005.

[8] H. Bruyninckx, "Open robot control software: The orocos project." in *IEEE/RAS Intl. Conf. on Robotics and Automation*, vol. 3, 2001, pp. 2523–2528.

[9] O. Stasse, B. Verrelst, P.-B. Wieber, B. Vanderborght, P. Evrard, A. Kheddar, and K. Yokoi, "Modular architecture for humanoid walking pattern prototyping and experiments," *Advanced Robotics, Special Issue on Middleware for Robotics –Software and Hardware Module in Robotics System*, vol. 22, no. 6, pp. 589–611, 2008.

[10] F. Kanehiro, K. Fujiwara, S. Kajita, K. Yokoi, K. Kaneko, H. Hirukawa, Y. Nakamura, and K. Yamane, "Open architecture humanoid robotics platform," in *International Conference on Robotics and Automation*, May 2002, pp. 24–30.

[11] P. Evrard, N. Mansard, O. Stasse, A. Kheddar, T. Schauss, C. Weber, A. Peer, and M. Buss, "Intercontinental, multimodal wide-range telecooperation using a humanoid robot," in *IEEE/RSJ Intelligent Conference on Intelligent Robots and Systems*, 2009, p. submitted.

[12] O. Stasse, F. Lamiraux, A. Kheddar, K. Yokoi, R. Ruland, and W. Prinz, "Integration of humanoid robots in collaborative working environment: A case study on motion generation," in *International Conference on Ubiquitious Robots and Ambient Intelligence*, 2008, pp. 211–216.

[13] W. Prinz, H. Loh, M. Pallot, H. Schaffers, A. Skarmeta, and S. Decker, "Ecospace - towards an integrated collaboration space for eprofessionals," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006.*, 2006, pp. 1–7.

[14] F. Lamiraux, J.-P. Laumond, C. V. Geem, D. Boutonnet, and G. Raust, "Trailer-truck trajectory optimization for airbus a380 component transportation," *IEEE Robotics and Automation Magazine*, vol. 12, no. 1, 2005.

[15] E. Yoshida, C. Esteves, I. Belousov, J.-P. Laumond, T. Sakaguchi, and K. Yokoi, "Planning 3-d collision-free dynamic robotic motion through iterative reshaping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1186–1198, 2008.

[16] N. Mansard and F. Chaumette, "Task sequencing for sensor-based control," *IEEE Trans. on Robotics*, vol. 23, no. 1, pp. 60–72, February 2007.

[17] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *International Journal of Robotics Research*, vol. 3, no. 1, pp. 43–53, Fevrier 1987.

[18] J. Park and O. Khatib, "Contact consistent control framework for humanoid robots," in *IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, Orlando, USA, Mai 2006.

[19] K. Doty, C. Melchiorri, and C. Bonivento, "A theory of generalized inverses applied to robotics," *Int. J. Robotics Research*, vol. 12, no. 1, pp. 1–19, Dcembre 1993.